TRABALHO DE GRADUAÇÃO

Aplicação de raciocínio formal para percepção e manipulação de objetos com o robô humanóide Nao

Por,

Felipe Moreira Ramos

Brasília, Dezembo de 2015

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Aplicação de raciocínio formal para percepção e manipulação de objetos com o robô humanóide Nao

Por,
Felipe Moreira Ramos

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Antônio Padilha L. Bó, ENE/UnB _____
Orientador

Prof. Mariana Costa B. Matias, FGA/UnB _____
Co-Orientadora

Prof. Guilherme Novaes Ramos, CIC/UnB _____
Examinador Interno

Brasília, Dezembo de 2015

FICHA CATALOGRÁFICA

FELIPE, MOREIRA RAMOS
Dissertação de trabalho de graduação,

[Distrito Federal] 2015.

ix, 51p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2015). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Robótica Humanóide
2.Representação de Conhecimento
3. Raciocínio Formal
4.KnowRob

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

RAMOS, F. M., (2015). Aplicação de raciocínio formal para percepção e manipulação de objetos com o robô humanóide Nao. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT. TG-$n°$12/2015, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 51p.

CESSÃO DE DIREITOS

AUTOR: Felipe Moreira Ramos

TÍTULO DO TRABALHO DE GRADUAÇÃO: Aplicação de raciocínio formal para percepção e manipulação de objetos com o robô humanóide Nao.

GRAU: Engenheiro ANO: 2015

Felipe Moreira Ramos

Endereço de e-mail: xfelipemoreirax@outlook.com

Dedicatória

Ao professor Antônio Padilha, pela paciência na orientação e incentivo que tornaram possível a conclusão deste trabalho de graduação. E à minha família, pelo apoio que me deram durante essa fase da minha vida.

Felipe Moreira Ramos

RESUMO

Desenvolver um comportamento de robô para aplicações de manipulação de objetos em ambientes para seres humanos é uma tarefa complexa. Ela exige conhecimento teórico em diversas áreas de pesquisa e um sistema capaz de representar as informações necessárias para o robô se adaptar às diversas situações possíveis. Este projeto tem como objetivo aplicar em um robô humanóide o uso de raciocínio formal para realizar a tarefa de procurar a manipular objetos em um ambiente simplificado a partir de instruções especificadas pelo usuário. Foi desenvolvido uma plataforma de pesquisa baseada em ROS para integrar os diferentes módulos do sistema, que incluem percepção de objetos, controle do robô, interface com o usuário, representação de conhecimento e tomada de decisão por raciocínio formal. Para o desenvolvimento do projeto foram utilizados um robô NAO e um sensor Kinect, além de marcadores padronizados para representar os objetos no ambiente. Os experimentos feitos durante o projeto serviram para verificar o funcionamento de todos os módulos da plataforma de pesquisa e validar a aplicação desenvolvida. As tarefas de procurar e manipular o objeto foram testadas apenas de forma separada. Mesmo com as limitações do robô, foi possível desenvolver uma manipulação simples através das informações obtidas pelo sistema de percepção. Os experimentos da busca por objetos mostraram a capacidade do sistema de seguir diferentes instruções do usuário e encontrar o objeto alvo utilizando o conhecimento adquirido e adaptando as ações para diferentes circunstâncias.

Palavras Chave: robótica humanóide, representação de conhecimento, raciocínio formal, KnowRob

ABSTRACT

Developing a robot behavior for object manipulation applications in human-scale enviroments is a complex task. It demands theoretical knowledge about many fields of research and a system capable of represent the required informations for the robot to adapt to different possible scenarios. This project aims to apply in a humanoid robot the use of formal reasoning for searching and manipulating objects in a simplified environment according to instructions specified by the user. It was developed a research platform based on ROS to integrate different modules of the system, which includes object perception, robot control, an interface with the user, knowledge representation and decision making by formal reasoning. For the project development, it was used a robot NAO and a sensor Kinect, as well as standardized markers to represent the objects in the enviroment.

The experiments done during the project served to verify the operation of all modules of the research platform and validate the developed application. The searching e manipulation tasks were tested separately only. Even with the limitations of the robot, it was possible to develop a simple manipulation using the information obtained through perception system. The experiments of searching an object showed the ability to follow different instructions from the user and find the target object using the aquired knowledge and adapting the actions to different circumstances.

Keywords: humanoid robotics, knowledge representation, formal reasoning, KnowRob

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

Introdução

1.1 Contextualização

Um dos maiores desafios atuais da robótica é a realização de tarefas de manipulação em ambientes para seres humanos. Com a evolução dos *hardwares,* sistemas de percepção e de controle, estão surgindo cada vez mais robôs capazes de executar tarefas *pick-and-place* e até mesmo manipulações mais complexas, como cozinhar. Entretanto, realizar tarefas em um ambiente dinâmico como casas e hospitais ainda é um trabalho desafiador, sendo que frequentemente os robôs não têm o conhecimento para executar essas tarefas de forma correta [1].

Principalmente quando o robô recebe uma instrução informal, é necessário uma grande quantidade de conhecimento para entender corretamente. Idealmente, o usuário deveria ser capaz de explicar uma tarefa para o robô da mesma forma que ele explicaria para um ser humano. Mas tais descrições geralmente carecem de muita informação que são óbvias para as pessoas. Seres humanos conseguem entender instruções incompletas por causa da sua notável capacidade de guardar uma grande quantidade de conhecimento e recuperar rapidamente tudo que ele precisa saber para a dada situação. Como uma pessoa pode assumir que as outras pessoas têm esse tipo de conhecimento, ela não precisa explicar detalhadamente como realizar uma tarefa ou como usar um objeto.

Comparados aos seres humanos, os robôs ainda carecem desse tipo de capacidade. Normalmente, eles seguem passo a passo planos pré-programados para executar tarefas sem ter a noção dos efeitos que suas ações têm, porque elas precisam ser feitas, e o que são os objetos com os quais está integarindo. O robô só sabe que ele precisa fazer algo, mas não sabe por que, que outras opções existe para alcançar o mesmo objetivo, ou como adaptar uma ação para diferentes situações.

Utilizar uma representação formal de conhecimento que está separada do controle do robô permite que esses aspectos sejam mais explícitos e aumenta a flexibilidade e reutilização das informações. Ao adicionar novos conhecimentos, não é necessário mudar o controle do robô. A forma como ele acessa as informações não muda, mas os resultados obtidos são baseados no novo conhecimento expandido. Dependendo das novas informações adicionadas, podem surgir novas soluções ou restrições nos resultados. A representação formal também permite que um conhecimento já descrito seja utilizado múltiplas vezes. Como por exemplo, as propriedades de um objeto podem

ser usadas para reconhecê-lo, inferir onde procurá-lo, ou parametrizar as ações para interagir com esse objeto específico.

Como o conhecimento está separado do sistema principal, ele pode facilmente ser utilizado em diversos contextos. Como exemplo, pode-se considerar um cenário onde um robô doméstico deve entregar um sanduíche para uma pessoa [2]. Como o sanduíche pode estar fora do alcance do robô, ele precisa primeiro procurar por ele para depois realizar a tarefa. Durante a busca, o robô precisa raciocinar sobre que lugares dentro de um ambiente de larga escala é mais provável que ele encontre um saduíche enquanto considera informações úteis sobre esses lugares, como o custo de locomoção. Ou seja, o robô deve ir para o local que maximiza as informações obtidas para encontrar o sanduíche. Por exemplo, o robô pode inferir que é mais provável encontrar um sanduíche em um restaurante *fast-food*. Entretanto, como o restaurante é muito longe, o robô decide procurar primeiro na cozinha. A cozinha tem uma menor probabilidade de sucesso, mas está mais próxima da posição atual. Ao chegar no local, o robô utiliza um sistema de percepção para encontrar o sanduíche e planeja a trajetória para pegá-lo e entregar à pessoa.

1.2 Definição do problema

Para que o robô realize tarefas de manipulação em ambientes complexos e dinâmicos, é necessário considerar diversas situações [1]:

1. O robô deve entender instruções dadas por humanos e traduzi-lás para especificações da tarefa a ser realizada, determinando que informações estão faltando e como adicionar ações e descrições de objetos que não fazem parte da descrição original;
2. Dependendo do contexto, as ações precisam ser adaptadas com os parâmetros corretos. Pode ser necessário adicionar restrições ao movimento do robô ou mudar que objeto deve ser manipulado;
3. Geralmente, as decisões não podem ser tomadas antes que o plano de ações seja executado, pois elas dependem do contexto atual e o estado de crença do robô. Por isso, o robô deve interpretar controles gerais em que a tomada de decisão não foi totalmente codificada, mas precisa ser inferida durante a execução.

Além disso, é importante empregar conhecimento semântico para limitar o espaço de busca. Embora os sistemas de percepção desempenhem um papel fundamental na busca de objetos dentro da robótica, a percepção é computacionalmente cara, e se o robô não sabe onde pode encontrar o objeto, este só será encontrado através de um exaustivo método de busca ou por acaso [2].

1.3 Objetivos do projeto

Este projeto tem como objetivo implementar em um robô humanóide o uso de raciocínio formal para realizar a tarefa de procurar e manipular objetos em um ambiente simplificado. Esta aplicação

servirá como validação para no futuro desenvolver tarefas mais complexas em ambientes dinâmicos. Para isso, é necessário uma representação formal do conhecimento para raciocinar sobre instruções com pouca informação, decidir sobre o local em que pode encontrar o objeto e ajustar os parâmetros das ações de acordo com as circunstâncias.

Na aplicação, o usuário indicará dentre os objetos que fazem parte do ambiente simplificado qual o robô deve manipular. Para encontrar o objeto especificado, o robô deverá procurar por uma sequência de objetos que estão entre o objeto que o robô está vendo e o que ele deve manipular. A sequência será encontrada através do mapeamento do ambiente que utiliza as informações da base de conhecimento.

Como sistema principal, será desenvolvido uma plataforma de pesquisa que integra a base de conhecimento com o robô humanóide NAO e um sistema de percepção baseado em duas câmeras, uma com visão completa do ambiente utilizado e a outra do próprio robô.

1.4 Apresentação do manuscrito

Este trabalho está dividido em capítulos. O capítulo 1 contextualiza o tema do trabalho e introduz o problema a ser resolvido e os objetivos do projeto. O capítulo 2 explica os fundamentos teóricos e discorre sobre as ferramentas utilizadas. O capítulo 3 mostra o desenvolvimento da plataforma de pesquisa, os componentes necessários para o seu funcionamento e o desenvolvimento da aplicação de raciocínio formal em um robô humanóide. O capítulo 4 explica sobre os experimentos realizados e os seus resultados. O capítulo 5 conclue o trabalho e fala sobre trabalhos futuros.

Capítulo 2

Fundamentação

2.1 Introdução

Neste capítulo, são apresentados os fundamentos teóricos e ferramentas utilizados neste trabalho. Com base no objetivo principal do trabalho apresentado no Capítulo 1, foi necessário realizar uma pesquisa bibliográfica sobre controle de robô humanóide, *Semantic Web*, raciocínio por inferência e detecção de objetos.

Este trabalho foi baseado principalmente na busca de objetos em ambientes de larga escala feito por Kunze at al. [2]. Eles fizeram experimentos onde o robô tinha que encontrar diferentes objetos, como copos e sanduíches, utilizando várias estratégias em um contexto de tarefas do tipo buscar e entregar dentro de uma construção com vários andares. A base de conhecimento que o robô utilizava foi desenvolvida com base na biblioteca `KnowRob` [3].

Para a detecção de objetos, este trabalho utiliza a biblioteca `ARToolKit` [4]. Ela é capaz de obter de forma precisa as coordenadas de um objeto no ambiente a partir de marcadores padronizados observados por uma câmera. Além disso, tanto o `KnowRob` como o `ARToolKit` possuem pacotes no *Robot Operating System (ROS)* [5], que além de ter várias ferramentas e bibliotecas, facilita a comunicação entre estes e outros pacotes usados para controlar o robô e o sistema de percepção.

2.2 Nao

O Nao é um robô humanóide desenvolvido pela empresa Aldebaran [6]. Ele possui 25 graus de liberdades (DoF, do termo em inglês), distribuidos entre os seus quatro membros e a cabeça. Além dos atuadores, o seu *hardware* contém oito sensores de pressão, duas câmeras, quatro microfones, dois emissores e receptores infravermelho, uma unidade de medida inercial, dentre outros.

O robô possui um sistema operacional dedicado baseado em Linux, o `Naoqi`. Este sistema operacional permite a interpretação dos dados obtidos pelo sensores e enviar comandos aos atuadores. Existe uma versão para ROS do `Naoqi`. As aplicações podem ser desenvolvidades tanto de forma embarcada, como através de um computador remoto. Dentre as aplicações que utilizam o robô Nao

podem ser citados interação homem-robô, localização e mapeamento, coreografia de movimentos e manipulação de objetos.

2.3 Kinect

O Kinect é um sensor de movimentos que foi desenvolvido para *Xbox 360* [7]. Devido ao seu custo benefício, ele também é utilizado em pesquisas de diversas áreas, como mapeamento, reconstrunção de cenas, reconhecimento de objetos e de pessoas. Ele possui uma câmera RGB, um sensor de profundidade e um processador interno.

A maior vantagem do uso deste sensor em uma aplicação é o seu mapeamento em profundidade, onde cada pixel possui a profundidade na cena real além da cor e intensidade na imagem. O mapeamento, por exemplo, permite a detecção de poses tridimensionais de uma pessoa [8]. Para obter as imagens das câmeras do sensor, além do próprio *kit* de desenvolvimento, pode-se usar a biblioteca `OpenNI`, que possui uma versão no ROS.

2.4 Arquitetura de software para robótica

Um importante desafio na robótica diz respeito à necessidade de se integrar no *software* diferentes módulos do sistema. Entre as opções disponíveis, existe o ROS, um *framework* flexível feito para o desenvolvimento de *softwares* para robôs [5]. Ele foi desenvolvido para operar em geral no Ubuntu, um sistema operacional Linux, mas também é possível utilizá-lo em outros sistemas operacionais ou instalar diretamente em robôs compatíveis.

O ROS é uma coleção de ferramentas, bibliotecas e convenções que visa simplificar a tarefa de criar comportamentos complexos e robustos para uma grande variedade de plataformas robóticas. Ele foi construído para encorajar o desenvolvimento de *sotwares* para robôs de forma colaborativa, onde cada pesquisador ou laboratório contribui nas áreas que são especializados.

Uma das principais características do ROS é permitir a troca de informações em tempo real entre os nós, como são chamados os *softwares* desenvolvidos. Isso é feito através de tópicos e serviços [5]. A Figura 2.1 mostra um exemplo de comunicação entre nós. Tópicos permitem que um nó envie mensagens para o núcleo do ROS e outro nó receba esta mensagem quando for necessário. Os arquivos de mensagens para tópicos são padronizados. Cada linha possui um parâmetro com o tipo de valor que ele possui e o nome identificador. É possível inclusive que o tipo seja outra mensagem.

Serviços permitem que um nó cliente solicite informações para um nó servidor. Os arquivos de mensagens para serviços seguem o mesmo padrão que os tópicos, mas os parâmetros são divididos em duas partes: parâmetros do pedido e parâmetros da resposta. O nó cliente pode enviar na mensagem parâmetros necessários para o nó servidor criar a informação desejada. O nó servidor então retorna para o nó cliente os parâmetros especificados nos parâmetros de resposta.

A troca de mensagens ocorre através do núcleo do ROS. Este nó especial armazena as mensagens

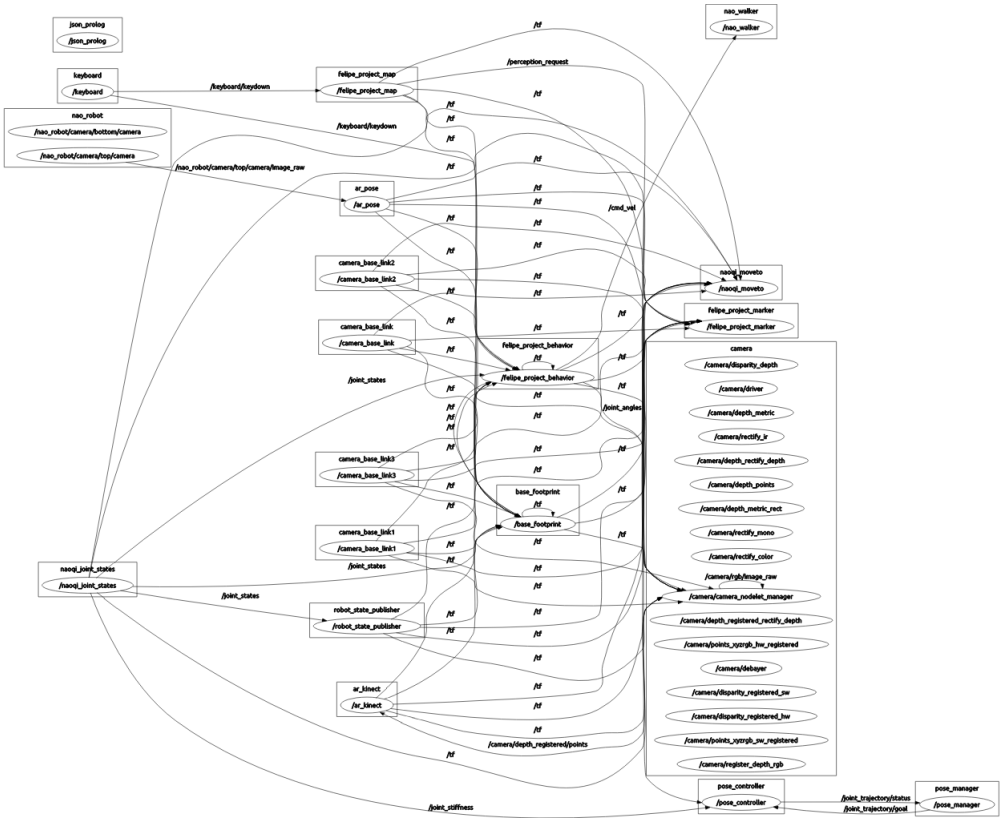


Figura 2.1: Vizualização da comunicação entre módulos usando a ferramente *rqt_ graph*. É mostrado a comunicação entre os módulos `Keyboard`, `OpenNI`, `ARToolKit`, `KnowRob`, `Naoqi`, sistema de percepção, controle do robô e o módulos específicos da aplicação.

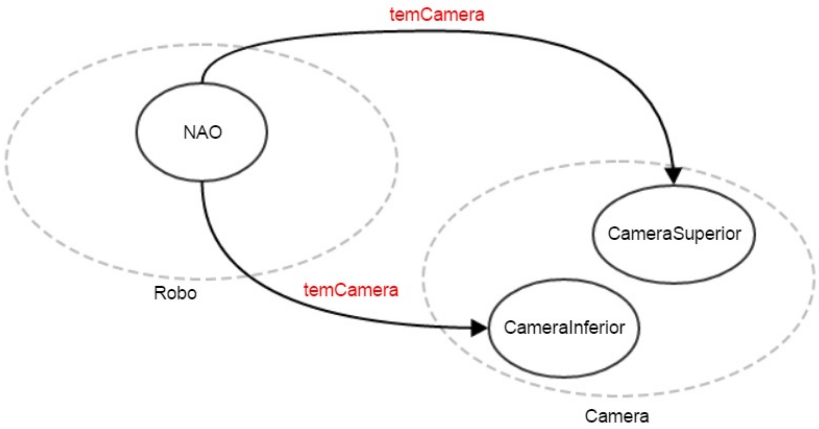


Figura 2.2: Exemplo de uma base de conhecimento. As elípses tracejadas são conceitos e as elípses internas são os indivíduos que pertencem ao conceito. Os arcos simbolizam as regras que relacionam um indivíduo a outro.

e envia para os nós que pediram pela informação. Isso permite que a taxa de envio e recebimento das mensagens de cada nó seja independente e que qualquer nó possa utilizar um tópico ou um serviço quando for necessário. Esta característica também é importante no teste de cada nó separadamente, pois é possível enviar mensagens com valores de teste ou criar um simples nó que identifica se a mensagem foi recebida.

2.5 Representação de conhecimento

Para que uma pessoa e uma máquina consigam se comunicar de forma intuitiva e fácil de ler ou ouvir, é necessário que a máquina entenda o significado preciso das informações transmitidas. Para isso, a máquina deve organizar e formalizar o conhecimento a fim de combinar e recuperar de forma inteligente a informação armazenada. No campo da *Semantic Web,* linguagens têm sido desenvolvidas com o objetivo de prover informações na *Web* em uma forma suficientemente formal e estruturada para que humanos e máquinas possam entender [9].

Semantic Web é uma aplicação das linguagens de lógica de descrição (DLs, do inglês *Description Logics).* As DLs são focadas em resolver problemas de decisão para representar conhecimento e realizar raciocínio formal por inferência ou por dedução. Uma classe de problemas é dita de decisão quando há um algoritmo genérico que provê uma resposta do tipo sim-ou-não em tempo finito para um conjunto de afirmações lógicas que tenta inferir outra afirmação.

Todas as DLs são baseadas em um vocabulário contendo indivíduos, conceitos e regras. Com estes vocabulários são construídas bases de conhecimento, que possuem o conhecimento necessário para resolver o problema. Um exemplo de base de conhecimento com indivíduos, conceitos e regras é mostrado na Figura 2.2. Usualmente, essas bases de conhecimento são divididas em parte assertiva, chamada de *ABox* e parte terminológica, chamada de *TBox.*

Na *ABox,* as afirmações podem ser de conceito, indicando que um indivíduo pertence a um

determinado conceito ou de regras, que relacionam um indivíduo a outro através de uma regra. Já a *TBox* possui afirmações universais. Entre elas pode-se afirmar que um conceito herda de outro conceito ou que todos os indivíduos pertencentes a um conceito possuem uma afirmação de regra.

2.5.1 KnowRob e OWL

É possível representar o conhecimento no ROS utilizando *Semantic Web* com o *software* `KnowRob`. `KnowRob` é um sistema de processamento de conhecimento que combina representação de conhecimento, métodos de raciocínio formal, técnicas para aquisição de conhecimento e para relacionar esse conhecimento a um sistema real [3]. Ele combina conhecimento enciclopédico estático, conhecimento de senso comum, descrição de tarefas, modelos de ambiente, informação de objetos e informação sobre ações observadas.

O conhecimento no `KnowRob` pode ser adquirido por diversas fontes: da *Web*, escrito manualmente, percebido pelo robô e derivado de observações dos seres humanos. A representação do conhecimento é feita usando a liguagem de *Semantic Web* mais difundida, a Web Ontology Language (OWL) [9]. Os métodos de raciocínio formal para inferir novas informações usam Prolog, uma linguagem de programação lógica de propósito geral [10].

A OWL é uma linguagem usada para descrever entidades que são inerentes de documentos da Web e aplicativos [11]. Ela é uma extensão do *Resource Description Framework* (RDF) e seu principal objetivo é criar ontologias, descrições dos tipos de entidades que existem no mundo observado e como eles se relacionam.

Devido à semântica formal, é possível inferir consequências lógicas sobre as entidades, isto é, fatos que não estão explicitamente na ontologia. Este raciocínio formal feito na OWL é baseado na *suposição de mundo aberto*. Isto significa que não se pode assumir que algo não existe até que seja estabelecido que ele não existe. Em outras palavras, só porque algo não foi estabelecido como verdadeiro, não implica que ele é falso.

Uma ontologia na OWL consiste de indivíduos, classes (conceitos) e propriedades (regras). Indivíduos representam os objetos no mundo observado, enquanto que classes são conjuntos de indivíduos. As classes são descritas como os requisitos necessários para um indivíduo ser membro da classe. Se um indivíduo não possui os requisitos, ele não pode ser uma instância da classe. As classes podem ser organizadas em uma hierarquia, onde os descendentes de uma classe herdam os requisitos da classe pai.

Propriedades são relações binárias entre indivíduos. Uma propriedade relaciona indivíduos que pertencem ao seu domínio a indivíduos que pertencem ao seu intervalo. O domínio e o intervalo podem ser uma classe ou a intercecção de várias classes. Propriedades podem ter características específicas, como ser inversa, transitiva, simétrica ou funcional [11]. Além disso, elas podem ser usadas como restrições para que um indivíduo pertença a uma classe. Ou seja, o indivíduo precisa participar de relações que usam essas propriedades. Uma restrição pode ser quantificadora, cardinal ou de valor [11].

O tipo mais comum de restrições utilizadas é o quantificador, que pode ser dividido em restrições

existenciais e restrições universais. As restrições existenciais descrevem classes onde um indivíduo precisa ter pelo menos uma das restrições para fazer parte da classe. As restrições universais descrevem classes onde um indivíduo deve ter apenas uma dessas restrições para fazer parte da classe. Com a ajuda de um algoritmo de raciocínio formal, é possível inferir quais indivíduos pertencem a uma classe e construir uma hierarquia entre as classes.

No KnowRob, o conhecimento obtido dos arquivos OWL é armazenado como triplas [12]. Essas triplas possuem como elementos um sujeito, um predicado e um objeto. Predicados são propriedades do indivíduo sujeito que o relacionam ao indivíduo objeto. Um sujeito pode ter vários predicados e vários objetos relacionados a ele. A classe de um indivíduo é relacionada a ele como se fosse um objeto através do predicado especial *rdf:type.*

2.5.2 Prolog

Prolog é uma linguagem para Programação Lógica [10]. Está além do escopo deste texto explicar todas as suas funcionalidades. O foco reside então nos elementos básicos do Prolog necessários para o entendimento deste trabalho. Primeiramente são descritos seus três construtores básicos: fatos, regras e quetionamentos. Em seguida são apresentados elementos e operações mais específicos, como operações de conjunção e disjunção, recursividade, listas e operação *if-then-else*. Vários exemplos são utilizados para ilustrar como a linguagem funciona.

2.5.2.1 Fatos, regras e quetionamentos

Fatos são usados para estabelecer algo como incondicionalmente verdade, enquanto que regras estabelecem informações que são condicionalmente verdade no domínio de interesse. Uma coleção de fatos e regras formam uma base de conhecimento. Programar em Prolog consiste em escrever bases de conhecimento e impor quetionamentos. Um quetionamento é o processo de adquirir informação armazenada. Considere a seguinte base de conhecimento.

```
temMao(robo(nao)).
agarraPeca(X) :- temMao(robo(X)).
```

A primeira cláusula é um fato e a última é uma regra. A regra significa que "Se o robô X tem mão, então ele pode agarrar uma peça".

Os nomes com a primeira letra minúscula são conhecidos como átomos e os nomes com a primeira letra maiúscula são variáveis. Quando um átomo está seguido de outro átomo ou variável dentro de parêntesis, diz-se que ele é um predicado e o que está dentro do parêntesis são seus argumentos. Um predicado pode ser escrito como predicado/N, onde N é seu número de parâmetros.

Variáveis são inicialmente indefinidas e podem ser associadas a um átomo ou predicado. Isso significa que elas têm o mesmo significado que o átomo ou predicado. Pode-se adquirir informaçãos sobre a base de conhecimento impondo um quetionamento como a seguir:

```
?- agarraPeca(nao).
true.
```

true é a resposta do Prolog quando a informação é verdadeira. Prolog é capaz de inferir informações que não estão explicitamente na base de conhecimento, como mostrado neste exemplo. Nele, após encontrar o predicado *agarraPeca(X)* na base de conhecimento, o Prolog associa a variável X ao átomo *nao* e então procura pela condição que torna esse fato verdadeiro, *temMao(robo(nao)).* Em seguida, ele encontra o fato *temMao(robo(Y))* e associa a variável Y a *nao* para finalmente encontrar que o fato *robo(nao)* é verdadeiro, inferindo que *agarraPeca(nao)* também é verdadeiro.

É possível usar variáveis ao impor um quetionamento, como mostrado a seguir:

```
?- agarraPeca(X).
X = nao;
false.
```

Prolog irá encontrar todos os átomos e predicados que fazem o quetionamento ser verdadeiro. A resposta *false* indica que não há outras soluções.

2.5.2.2 Conjunção e Disjunção

As operações de conjunção são representadas em Prolog pelo símbolo ",". Elas podem ser utilizadas quando uma regra possui mais de um item à direita de ":-" e todos precisam ser verdadeiros para a regra ser verdadeira. As operações de disjunção são representadas em Prolog pelo símbolo ";". Elas podem ser utilizadas quando uma regra possui mais de um item à direita de ":-" e apenas um precisa ser verdadeiro para a regra ser verdadeira. O símbolo "," também é utilizado para separar os argumentos de um predicado. Considere a seguinte base de conhecimento:

```
humano(joao).
temBracos(robo(nao)).
temPernas(robo(nao)).
humanoides(X,Y) :-
   (temBracos(robo(X)),temPernas(robo(X)),humano(Y));
   (temBracos(robo(Y)),temPernas(robo(Y)),humano(X)).
```

A regra desta base de conhecimento diz que *"humanoides(X,Y)* é verdadeiro se *humano(Y)* e temBracos(robo(X)) e temPernas(robo(X)) for verdadeiro ou *humano(X)* e temBracos(robo(Y)) e temPernas(robo(Y)) for verdadeiro". Utilizando variáveis em um quetionamento, a resposta será:

```
?- humanoides(X,Y).
X = nao,
Y = joao;
X = joao,
Y = nao;
    false.
```

Os valores das variáveis de uma solução são separados por ",", enquanto que diferentes soluções são separadas por ";". Conjunção e disjunção também são usados em quetionamentos para perguntar várias informações ao mesmo tempo.

2.5.2.3 Recursividade

Predicados podem ser definidos recursivamente. Em outras palavras, um predicado é definido recursivamente se uma ou mais regras de sua definição se referem a si mesmo. Para que a recursividade funcione corretamente, é necessário que o predicado tenha pelo menos duas cláusulas: uma cláusula base, que permite a parada da recursividade e outra que contém a recursão. Como o Prolog faz uma busca na base de conhecimento de cima para baixo [10], a cláusula base deve ser escrita antes da recursão. Caso contrário, a cláusula recursiva será sempre verificada primeiro e a busca nunca terminará.

No exemplo a seguir, é possível encontrar com a regra se um elemento Y está acima de outro elemento X quando há elementos Z entre eles. Note que para o quetionamento *acimaDe(marcador1, Y), Y* será associado a apenas *marcador3.* Isso ocorre porque a regra entende que um elemento está acima de outro se há apenas um único elemento entre eles.

```
logoAcimaDe(marcador4, marcador5).
logoAcimaDe(marcador3, marcador4).
logoAcimaDe(marcador2, marcador3).
logoAcimaDe(marcador1, marcador2).
acimaDe(X,Y) :-
    logoAcimaDe(X,Z),
    logoAcimaDe(Z,Y).
```

Refazendo a base de conhecimento utilizando recursividade, é possível encontrar se um elemento está acima de outro independente de quantos elementos estejam entre eles, como mostrado a seguir:

```
logoAcimaDe(marcador4, marcador5).
logoAcimaDe(marcador3, marcador4).
logoAcimaDe(marcador2, marcador3).
logoAcimaDe(marcador1, marcador2).
acimaDe(X,Y) :- logoAcimaDe(X,Y).
acimaDe(X,Y) :-
    logoAcimaDe(X,Z),
    acimaDe(Z,Y).
```

A resposta para o mesmo quetionamento agora possui como solução todos os níveis acima de *marcador1.*

```
?- acimaDe(marcador1,Y).
Y = marcador2 ;
Y = marcador3 ;
```

```
Y = marcador4 ;
Y = marcador5 ;
false.
```

2.5.2.4 Listas

Listas são conjuntos finitos de elementos. Elas são especificadas no Prolog delimitando os elementos com colchetes e separando-os com vírgula. Os elementos de uma lista podem ser qualquer tipo de objeto do Prolog, desde átomos, variáveis até outras listas.

Listas que não estão vazias podem ser divididas em duas partes: *head* e *tail*. *head* são os primeiros elementos da lista, enquanto que *tail* é a lista formada pelos elementos que sobraram. Para acessar a *head* e a *tail* é utilizado o operador '|', como mostrado a seguir.

```
?- [First, Second | Tail] = [abaixo, esquerda,
esquerda, acima].
First = abaixo,
Second = esquerda,
Tail = [esquerda, acima].
```

É possível acessar e manipular os elementos de listas utilizando predicados com recursividade. Alguns dos predicados mais utilizados são *member,* que verifica se um elemento faz parte de uma lista e *reverse,* que inverte a ordem dos elementos na lista. As cláusulas que formam esses dois predicados são mostradas a seguir.

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
accReverse([H|T],A,R) :- accReverse(T,[H|A],R).
accReverse([],A,A).
reverse(L,R) :- accReverse(L,[],R).
```

2.5.2.5 *If-then-else*

Prolog possui um predicado especial que permite testar uma cláusula e decidir o que fazer caso a resposta seja positiva ou negativa. Esse predicado é escrito como *(A -> B; C)*. Para o Prolog, isso significa: Se você pode provar *A*, então teste *B* e ignore *C*. Caso *A* falhe, teste *C* e ignore *B*.

Como mostrado no exemplo a seguir, esse predicado é importante quando se está criando e atualizando indivíduos de uma ontologia.

```
existe(Individuo) ->
    atualizar(Individuo);
    criar(Individuo).
```

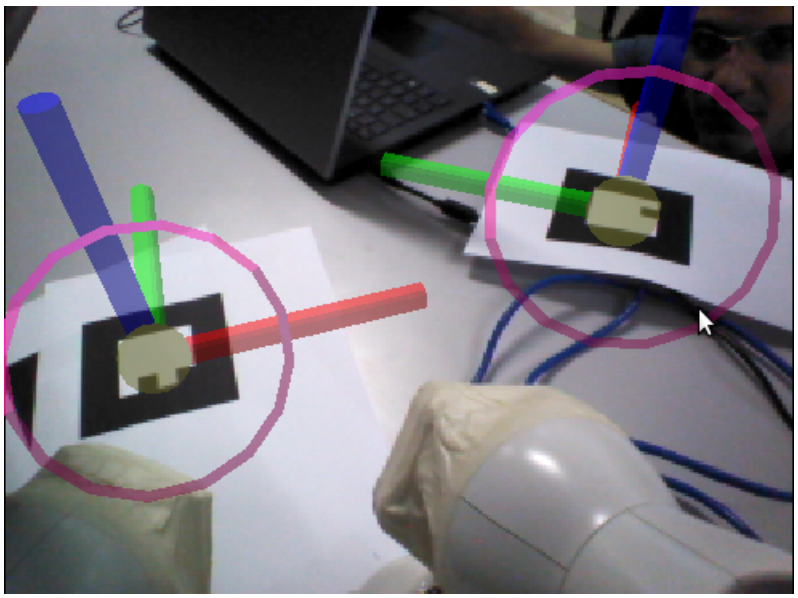


Figura 2.3: Detecção de marcadores. Os eixos azul, verde e vermelho representam as coordenadas do marcador no mundo em relação à câmera.

Se já existe na base de conhecimento um indivíduo identificado pelo átomo associado a *Individuo,* as suas informações serão atualizadas. Caso não exista, o indivíduo será criado e adicionado à base de conheicmento.

2.6 Detecção de objetos

Em aplicações onde um robô interage com objetos, é necessário detectar a posição e orientação do objeto em tempo real. Além disso, quando o objeto detectado é conhecido, outras informações como cor e formato podem ser adicionadas a uma base de conhecimento.

Uma maneira simples de estimar a posição e orientação em tempo real envolve o uso de marcadores. Os marcadores são imagens padronizadas, sendo que o sistema é treinado para detectar os padrões de cada uma. A utilização deles possibilita a detecção de objetos com características distintas de forma rápida e precisa. Um exemplo da detecção de marcadores pode ser visto na Figura 2.3.

O *software* `ARToolKit` é uma biblioteca com pacotes no ROS que facilita o desenvolvimento de aplicações de Realidade Aumentada (RA) [4]. RA é a sobreposição no mundo real de imagens gráficas criadas em computador, e tem muitas aplicações na indústria e em pesquisas acadêmicas. Outros usos da biblioteca são a detecção de objetos em ambientes reais e mapeamento do ambiente.

Uma das partes mais difíceis do desenvolvimento de aplicações de RA é calcular o ponto de vista do usuário em relação aos objetos do mundo real. O `ARToolKit` usa técnicas de visão computacional para calcular a posição e orientação reais de marcadores padronizados relativos a uma câmera. Cada marcador é identificado por um ID e possui como parâmetros o tamanho de um lado e o seu centro.

O objetivo do algoritmo utilizado pelo `ARToolKit` é encontrar a matriz de transformação T_{cm} apresentada na Equação 2.1. Ela transforma as coordenadas de um ponto no marcador em relação ao seu centro para coordenadas em relação à câmera.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} & V_{13} & W_x \\ V_{21} & V_{22} & V_{23} & W_y \\ V_{31} & V_{32} & V_{33} & W_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} = T_{cm} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} \tag{2.1}$$

Depois de aplicar o limiar na imagem de entrada, são extraídas as regiões cujo contorno pode ser delineado por quatro segmentos de reta. Os parâmetros (a, b, c) da Equação 2.2 para esses quatro segmentos de reta, junto com as coordenadas dos quatro vértices formados pelas suas intersecções são armazenados para processamento posterior.

$$ax + by + c = 0 \tag{2.2}$$

As regiões extraídas são normalizadas e comparadas com os marcadores especificados pelo usuário. A comparação é feita por padrões extraídos das imagens dos marcadores. Para realizar a normalização, é usada a Equação 2.3, que transforma as coordenadas de um ponto em relação ao centro do marcador para coordenadas na imagem. As variáveis da matriz de transformação são determinadas usando as coordenadas dos quatro vértices encontrados na imagem (x_c, y_c) e as coordenadas dos quatro vértices em relação ao centro do marcador a partir dos parâmetros especificados pelo usuário (X_m, Y_m).

$$\begin{bmatrix} hx_c \\ hy_c \\ h \end{bmatrix} = \begin{bmatrix} N_{11} & N_{12} & N_{13} \\ N_{21} & N_{22} & N_{23} \\ N_{31} & N_{32} & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ 1 \end{bmatrix} \tag{2.3}$$

Dada a matriz de projeção perspectiva $\mathbf{P}$ obtida pela calibração da câmera, ao substituir (x_c, y_c) da Equação 2.4 por (x, y) da Equação 2.2 para dois lados paralelos do marcador, obtém-se as equações dos planos que incluem os dois lados paralelos, representadas pela Equação 2.5.

$$P = \begin{bmatrix} P_{11} & P_{12} & P_{13} & 0 \\ 0 & P_{22} & P_{23} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} hx_c \\ hy_c \\ h \\ 1 \end{bmatrix} = P \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2.4}$$

$$aP_{11}X_c + (aP_{12} + bP_{22})Y_c + (aP_{13} + bP_{23} + c)Z_c = 0 \tag{2.5}$$

Considerando que os vetores normais de dois planos são n_1 e n_2 , o vetor direção de dois lados paralelos pertencentes a esses planos é dado pelo produto $n_1 \times n_2$. Ao comparar dois vetores unitários de direção obtidos de dois conjuntos de dois lados paralelos do marcador, esses vetores deveriam ser perpendiculares. Entretanto, erros do processamento da imagem fazem com que

eles não sejam exatamente perpendiculares. Baseado nesses vetores, dois novos vetores, v_1 e v_2, são criados de forma que sejam perpendiculares. Considerando um terceiro vetor unitário v_3 perpendicular aos dois primeiros, a componente de rotação $V_{3\times3}$ da matriz de transformação T_{cm} especificada por $\left[V_1^t\ V_2^t\ V_3^t\right]$.

Substituindo a componente de rotação $V_{3\times3}$, as coordenadas dos quatro vértices na imagem e em relação ao centro do marcador nas Equações 2.1 e 2.4, são obtidas oito equações com os elementos W_x,W_y e W_z da componente de translação da matriz de transformação T_{cm}. Os valores desses elementos podem ser obtidos a partir dessas equações. Com as componentes de rotação e translação encontradas, a matriz de transformação T_{cm} é então obtida.

2.7 Cinemática direta e inversa

O problema da cinemática de um robô manipulador consiste em descrever o seu movimento sem considerar as forças e torques que causam o movimento. A cinemática pode ser dividida em cinemática direta e cinemática inversa, que estão detalhadas a seguir.

2.7.1 Cinemática direta

Um robô manipulador é composto por uma série de *links* conectados por juntas [13]. Em geral, as juntas podem ser tanto de revolução como prismáticas. Considerando que cada junta tem apenas um grau de liberdade para se mover, a ação de cada uma pode ser descrita por um único número real; o ângulo de rotação no caso de junta de revolução ou o deslocamento no caso de junta prismática. O objetivo da análise por cinemática direta é determinar o efeito acumulativo de todo o conjunto de variáveis de junta. Isto é, ela determina a posição e orientação do último *link,* dados os valores das variáveis de junta. O último *link* é chamado de efetuador terminal.

Um robô manipulador com n juntas tem $n+1$ *links,* já que cada junta conecta dois *links.* As juntas são numeradas de 1 a n, e os *links* de 0 a n, começando pela base. Por convenção, a junta i, cuja variável é denotada por q_i, conecta o *link* $i-1$ ao *link* i.

Considere que A_i é a matriz de transformação homogênea que expressa a posição e orientação do sistema de coordenadas $o_ix_iy_iz_i$ em relação ao sistema de coordenadas $o_{i-1}x_{i-1}y_{i-1}z_{i-1}$. Ela não é constante, pois varia assim que a configuração do robô muda. Entretanto, assumindo que todas as juntas são de revolução ou prismática, A_i é uma função de uma única variável de junta q_i. Em outras palavras,

$$A_i = A_i(q_i). \tag{2.6}$$

A matriz de transformação homogênea que expressa a posição e orientação de $o_jx_jy_jz_j$ em relação a $o_ix_iy_iz_i$é chamada, por convenção, de matriz de transformação. Ela é denotada por

$$T_j^i = \begin{cases} A_{i+1}A_{i+2}...A_{j-1}A_j & , i < j \\ I & , i = j \\ (T_i^j)^{-1} & , j > i \end{cases} . \tag{2.7}$$

Assim, a posição e orientação do efetuador terminal em relação a base do robô é dada por

$$H = T_n^0(q_1, ..., q_n) = A_1(q_1)...A_n(q_n), \tag{2.8}$$

onde

$$H = \begin{bmatrix} R_n^0 & o_n^0 \\ 0 & 1 \end{bmatrix}. \tag{2.9}$$

Como mostrado na Equação 2.9, a posição do efetuador é dada pelo vetor de três dimensões o_n^0 e a orientação é dada pela matriz 3×3 de rotação R_n^0.

2.7.2 Cinemática inversa

O problema da cinemática inversa consiste em encontrar as variáveis de juntas em termos da posição e orientação do efetuador terminal. Em geral, ele é mais difícil que o problema da cinemática direta. Dado a transformação homogênea 4×4 da Equação 2.9, deve-se encontrar uma ou mais soluções para a equação

$$T_n^0(q_1, ..., q_n) = H, \tag{2.10}$$

onde

$$T_n^0(q_1, ..., q_n) = A_1(q_1)...A_n(q_n). \tag{2.11}$$

Aqui, H representa a posição e orientação desejada para o efetuador terminal. O objetivo é encontrar os valores para as variáveis de junta $q_1, ..., q_n$ tal que $T_n^0(q_1, ..., q_n) = H$.

A Equação 2.10 resulta em doze equações não lineares com n variáveis icógnitas, que podem ser escritas como

$$T_{ij}(q_1, ..., q_n) = h_{ij}, \quad i = 1, 2, 3, \quad j = 1, ..., 4, \tag{2.12}$$

onde T_{ij} e h_{ij} referem-se às entradas não triviais de T_n^0 e H, respectivamente.

Enquanto que o problema da cinemática direta sempre tem uma única solução que pode ser obtida simplesmente analisando as equações diretas, o problema da cinemática inversa pode ou não ter uma solução. E mesmo que uma solução exista, ela pode ou não ser única. Além disso,

devido a essas equações da cinemática inversa serem em geral complicadas funções não lineares das variáveis de junta, mesmo que elas existam as soluções podem ser difíceis de obter.

Para resolver o problema da cinemática inversa, é possível utilizar métodos para encontrar uma solução de forma fechada ou uma solução numérica. O pacote `IKFast` disponível no ROS encontra soluções da cinemática inversa utilizando um método numérico. Este pacote é capaz de gerar uma solução numérica para a cinemática inversa de um robô a partir de seu *Unified Robot Description Format*(URDF). Um URDF possui as informações sobre os *links* e juntas do robô. Como exemplo, no Capítulo 3 são apresentados os parâmetros para o robô humanóide NAO.

Capítulo 3

Desenvolvimento

3.1 Introdução

Neste capítulo, é apresentada uma visão geral do projeto. Em seguida é descrito o desenvolvimento da plataforma de pesquisa implementada em ROS com base em pacotes desenvolvidos por outros pesquisadores, como o `OpenNI`, `ARToolKit`, `KnowRob` e `Naoqi`. O desenvolvimento é dividido em sistema de percepção e controle do robô. No capítulo também é descrito um método padrão para desenvolver aplicações e as especificações de cada etapa da aplicação de procurar e manipular objetos especificados pelo usuário em um ambiente simplificado.

3.2 Visão geral do projeto

Para a implementação da plataforma de pesquisa e aplicação, foi utilizado um notebook dedicado, uma plataforma robótica Nao [6] e um sensor Kinect [7]. O notebook, além de realizar todo o processamento, é responsável pela comunicação do sistema com o Nao e o Kinect. As suas especificações estão na Tabela 3.1.

Especificações	Descrição
Modelo	Dell Inspiron 15 3542
Sistema Operacional	Ubuntu 12.04 64bits
Processador	Intel® Core™ i3-4005U, 1,7 GHz
Memória RAM	4 GB, DDR3, 1600MHz
Disco Rígido	500 GB
Placa de Vídeo	Intel® HD Graphics 4400 Integrada

Tabela 3.1: Especificações do notebook utilizado.

A versão do ROS instalada é a Hydro. Esta versão foi escolhida porque alguns dos pacotes utilizados só foram atualizados até essa versão, enquanto que outros pacotes são recentes e só funcionam a partir do Hydro. O Ubuntu 12.04 é o Sistema Operacional padrão para a versão

Hydro.

Os marcadores padronizados foram gerados por David Johnson at al. [14]. Eles são sequências de quadrados pretos em um fundo branco e bordas pretas que representam um número binário, como mostrado na Figura 3.1. Os cartões utilizados têm $0.102m$ ou $0.040m$ de lado.

Figura 3.1: Exemplo de cartão marcado. Os dois pequenos quadrados pretos representam o número binário 11, que equivale a 3 em número decimal.

O diagrama da Figura 3.2 mostra uma visão geral da plataforma de pesquisa. Os módulos são representados na segunda linha, enquanto que as entradas e saídas estão na terceira linha. O fluxo de informações ocorre da esquerda para a direita e toda a comunicação entre os módulos é feita através de mensagens no ROS.

Os pacotes desenvolvidos por outros pesquisadores são o `OpenNI`, `ARToolKit`, `KnowRob` e `Naoqi`. O que foi implementado é dividido em três partes: sistema de percepção, controle do robô e o código específico da aplicação.

O sistema de percepção utiliza as informações dos marcadores detectados para criar indivíduos na base de conhecimento. A forma que as informações são atualizadas depende do tipo de percepção, que pode usar ou apagar as informações antigas.

A aplicação desenvolvida utiliza os dados obtidos pelo sistema de percepção e dos arquivos de OWL e Prolog para decidir que ações o controle do robô deve usar. A sequência de ações é especificada de acordo com os comandos enviados pelo usuário.

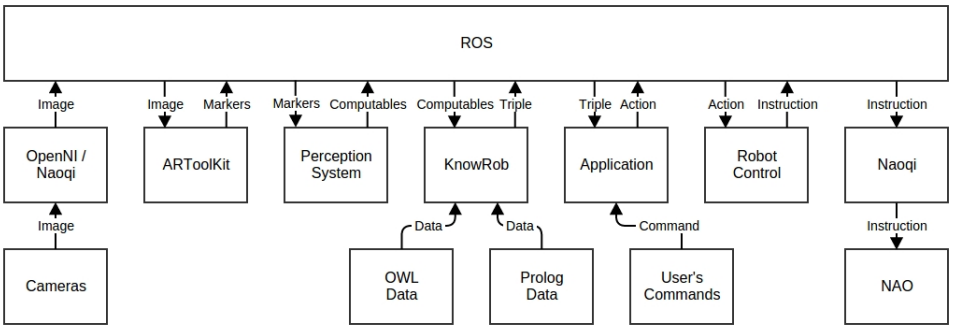


Figura 3.2: Diagrama do sistema completo. Dentre os módulos da segunda linha, foram desenvolvidos o *Perception System, Application* e *Control System.* As entradas e saídas estão na terceira linha. As comunicações entre os módulos são feitas por intermédio do *ROS* e o fluxo de informação é da esquerda para a direita.

3.3 Sistema de Percepção

O sistema de percepção é responsável pela detecção dos marcadores e por adicionar à base de conhecimento os indivíduos dos objetos ou atualizar as suas propriedades. A seguir, será explicada cada parte do sistema.

3.3.1 Detecção de marcadores

A detecção dos marcadores utiliza os pacotes do ROS `AR_Pose` e `AR_Kinect`. Estes pacotes são versões do `ARToolKit` para uma câmera normal e para o sensor Kinect, respectivamente. Para o seu funcionamento, é necessário criar um arquivo com as informações sobre cada marcador. Essas informações são o nome do padrão treinado a partir da imagem do marcador, um nome identificador, o comprimento de um lado em metros e a quantidade de marcadores que poderão ser detectados. Para facilitar a identificação, os nomes escolhidos para cada marcador devem ser os nomes das classes às quais os objetos pertencem.

3.3.2 Atualização da base de conhecimento

Para expandir a base de conhecimento e computar informações sob demanda, o `KnowRob` usa computáveis [3]. Os computáveis podem ser predicados associados a uma função do sistema de percepção que adquire as informações dos objetos. Além disso, pode-se computar relações qualitativas entre objetos, como a relação *inFrontOf-Generally*, que verifica se um indivíduo está a frente de outro.

Ao detectar um objeto, o sistema de percepção gera uma mensagem com a classe do objeto, a posição e a orientação em relação à câmera e a sua classe. Essas informações são utilizadas para atualizar as informações do indivíduo da base de conhecimento que representa o objeto detectado.

Cada objeto possui um nome identificador formado por sua classe e pelo tipo de percepção utilizado na sua detecção. Assim, um mesmo objeto pode ter diferentes indivíduos criados por diferentes câmeras.

Além das propriedades básicas de um indivíduo, é possível adicionar outras propriedades de acordo com a aplicação. Todas as informações são adicionadas à base de conhecimento utilizando um quetionamento. Há duas formas possíveis de adicionar essas informações: adicionar a última posição observada de cada objeto e adicionar apenas as informações mais recentes.

No primeiro caso, quando um novo objeto é detectado, as suas informações de posição, orientação e outras possíveis propriedades são atualizadas na base de conhecimento. Caso um objeto não seja detectado, as suas informações antigas permanecem. Esse método é utilizado principalmente para indivíduos que têm suas propriedades acessadas constantemente ou que têm uma posição fixa em relação a uma das câmeras. Na aplicação desenvolvida, o método é utilizado para criar o mapa semântico.

No segundo caso, a cada nova interação do sistema todos os objetos têm suas informações

antigas apagadas. Caso o objeto seja detectado, as informações são substituídas pelas novas, enquanto que se não for detectado, o seu indivíduo na base de conhecimento é apagado. Esse método é utilizado principalmente quando se quer saber quais objetos estão sendo detectados na interação atual do programa e quando a posição e orientação estão constantemente mudando. Na aplicação desenvolvida, o método é utilizado para o robô saber quais objetos sua câmera está detectando no momento enquanto ele se movimenta.

3.4 Controle do robô

O controle do robô é responsável por enviar para o robô os comandos que ele deve realizar com base nas ações especificadas pela aplicação. Os comandos são enviados utilizando a versão para o ROS do `Naoqi`, o *framework* que possui as bibliotecas necessárias para controlar o robô NAO. O controle desenvolvido permite realizar três tipos de ações distintas: locomoção, movimento dos braços e movimento da cabeça. A seguir, os tipos de ações são apresentados em detalhes.

3.4.1 Locomoção

O *framework* Naoqi já possui funções para realizar o controle da locomoção. É possível especificar uma posição alvo relativa à posição atual do robô ou a velocidade com que ele deve locomover.

3.4.2 Movimento dos braços

Para que o robô possa manipular objetos na aplicação desenvolvida, é necessário enviar os ângulos que posicionam a mão do robô onde o objeto está. A cinemática inversa é gerada pela solução numérica do pacote `IKFast` e o URDF do robô. Os parâmetros da hierarquia do braço esquerdo contidos no URDF são mostrados na Tabela 3.2. O elemento *base_link* é utilizado para que diferentes hierarquias do robô possuam a mesma referência.

No período de desenvolvimento deste projeto, a solução numérica do `IKFast` para 5 DoF não funcionou corretamente, causando problemas na orientação do efetuador terminal. Como cada braço do robô NAO possui 5 DoF, foi necessário adicionar mais um *link* em cada braço para poder usar a solução padrão de 6 DoF. Os *links* adicionados foram *l_gripper* para o braço esquerdo e *r_gripper* para o braço direito. Esses *links* correspondem aos *links* das mãos, que estão deslocados no eixo X em relação aos pulsos, considerando a convenção de Denavit-Hartenberg [13]. As mãos possuem atuadores para abrir e fechar os dedos, mas no arquivo URDF eles são representados como juntas de revolução. Assim, o ângulo calculado pela cinemática inversa pode ser somado ao ângulo do pulso como se fossem do mesmo grau de liberdade.

As coordenadas do objeto detectado são em relação à câmera e as coordenadas da mão são em relação à base do robô. Por isso, para saber a posição que a mão deve alcançar, as coordenadas do objeto devem ser transformadas para a base do robô. Isto é feito multiplicando as coordenadas

nome	índice	parente
base_link	0	
torso	1	*base_link*
LShoulder	34	*torso*
LBicep	35	*LShoulder*
LElbow	36	*LBicep*
LForeArm	37	*LElbow*
l_wrist	38	*LForeArm*
l_gripper	45	*l_wrist*

Tabela 3.2: Parâmetros do arquivo URDF para a hierarquia do braço esquerdo. O primeiro *link* é *torso*. O elemento *base_link* é utilizado para que diferentes hierarquias do robô possuam a mesma referência. Os índices são utilizados para idenfificar os *links*.

do objeto pela matriz de transformação da câmera em relação à base. Esta matriz está disponível no pacote `nao_description`.

Por possuir apenas 5 DoF no braço, o robô NAO não consegue segurar um objeto em todas as orientações possíveis. No caso do braço direito, a cinemática inversa é então calculada testando orientações com os eixos X e Y fixos e variando o eixo Z de 0^o a 90^o, considerando a convenção de Denavit-Hartenberg em relação ao *link* anterior. Isso permite que o robô agarre objetos em posições abaixo do ombro e na frente do botão de ligar, como mostrado na Figura 4.7.

Nem todas as soluções do `IKFast` ocorrem dentro da faixa de ângulos válidos para cada atuador. Antes de escolher uma solução, é verificado se todos os ângulos estão dentro dos limites mostrados na Tabela 3.3. Os limites do atuador *LElbowYaw* foram modificados para -0.50 e 0.50 para que o braço se movimente de fora para dentro na manipulação.

3.4.3 Movimento da cabeça

Como informado nas especificações do robô NAO, o campo de visão de suas câmeras é 60.9^o na horizontal e 47.6^o na vertical. A necessidade de criar uma ação para o movimento da cabeça surgiu porque o campo de visão não é suficiente para detectar objetos que estão em todas as regiões onde se pode agarrá-los. Essas regiões são abaixo dos ombros e próximo ao botão de ligar.

O movimento consiste em enviar os ângulos para os atuadores *HeadPitch* e *HeadYaw* da cabeça de forma que o objeto detectado esteja alinhado com o centro da câmera. A posição e orientação do objeto são obtidos em relação à câmera. Como visto na Figura 3.3, o sistema de coordenadas da câmera possui o eixo Z perpendicular ao plano da câmera e os eixos X e Y direcionados para a direita e para baixo na visão da câmera, respectivamente. Este sistema tem como referência o sistema de coordenadas da base do robô, posicionado no centro de massa. A orientação do sistema de coordenadas da base possui o eixo Z perpendicular ao lado superior do plano transversal do robô, o eixo Y perpendicular ao lado esquerdo do plano sagital e o eixo X perpendicular ao lado posterior do plano frontal.

Atuador	Mínimo(rad)	Máximo(rad)
HeadYaw	-2.09	2.09
HeadPitch	-0.67	0.51
LShoulderPitch	-2.09	2.09
LShoulderRoll	-0.31	1.33
LElbowYaw	-2.09	2.09
LElbowRoll	-1.54	-0.03
LWristYaw	-1.82	1.82
LHand	0.00	1.00
RShoulderPitch	-2.09	2.09
RShoulderRoll	-1.33	0.31
RElbowYaw	-2.09	2.09
RElbowRoll	0.03	1.54
RWristYaw	-1.82	1.82
RHand	0.00	1.00

Tabela 3.3: Limites dos atuadores do robô.

Os ângulos dos atuadores são calculados utilizando as Equações 3.1 e 3.2.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} - \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

$$HeadYaw = atan(y, x) \tag{3.1}$$

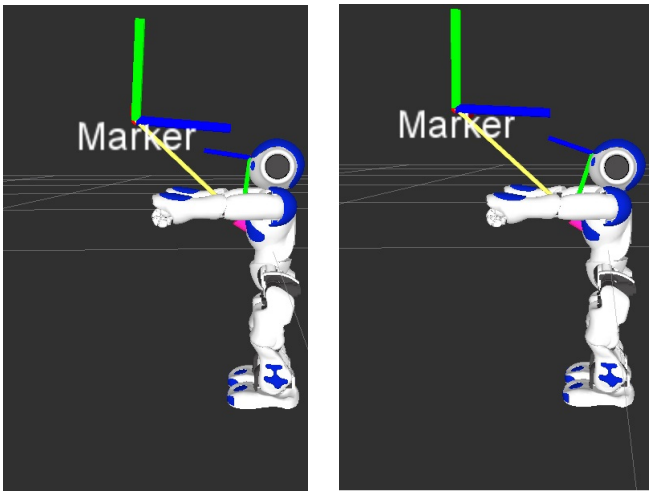
$$HeadPitch = -atan(z, x) \tag{3.2}$$

Como as coordenadas do objeto são em relação à câmera e as coordenadas da câmera são em relação à base do robô, as coordenadas do objeto devem ser transformadas para a base do robô. Isto é feito multiplicando as coordenadas pela matriz de transformação da câmera em relação à base disponível no pacote `nao_description`. Caso a transformação não seja feita, o alinhamento conterá erros, como mostrado na Figura 3.3.a. O alinhamento correto está na Figura 3.3.b.

3.5 Desenvolvimento da aplicação

A plataforma de pesquisa foi desenvolvida para facilitar a implementação de aplicações para robôs humanóides utilizando raciocínio formal. A plataforma já possui um sistema de percepção baseado no `ARToolkit`, controle do robô NAO e utiliza o `KnowRob` para processar o cohecimento.

Para implementar a aplicação, são necessárias quatro etapas, listadas a seguir.



(a) (b)

Figura 3.3: Alinhamento do objeto com a câmera. (a) mostra o alinhamento sem a transformação das coordenadas do objeto para a base do robô e (b) mostra o alinhamento com a transformação.

1. Criar arquivo OWL com as classes, propriedades e indivíduos utilizados na aplicação.
2. Criar arquivo Prolog com os predicados necessários para criar e acessar as informações da base de conhecimento.
3. Desenvolver interface com o usuário para transformar os comandos em quetionamentos.
4. Desenvolver comportamento do robô com base nas informações da base de conhecimento.

Os arquivos OWL e Prolog são utilizados na inicialização do `KnowRob` para gerar a base de conhecimento. Além de criar as próprias informações, é possível importar de outros arquivos que já vêm com o pacote. Os arquivos mais importantes utilizados neste trabalho são:

- `knowrob.owl` - ontologia com classificações genéricas de objetos, eventos e propriedades que restringem a classificação;
- `owl.pl` - conjunto de predicados utilizados para acessar as informações em OWL;
- `comp_spatial.owl` - conjunto de computáveis para relações espaciais;
- `comp_spatial.pl` - predicados associados aos computáveis para criar a relações espaciais entre objetos.

A aplicação desenvolvida para validar o trabalho foi procurar e manipular objetos especificados pelo usuário em um ambiente simplificado. O ambiente observável consiste de seis objetos, como mostrado na Figura 3.4. Eles são alinhados lado a lado para criar relações espaciais de acima, abaixo, à esquerda e à direita, que são chamadas de *aboveOf*, *bellowOf*, *leftOf* e *rightOf*, respectivamente. Há dois tipos de objetos - *Drink* e *Furniture*. Para simular uma situação real, o robô só

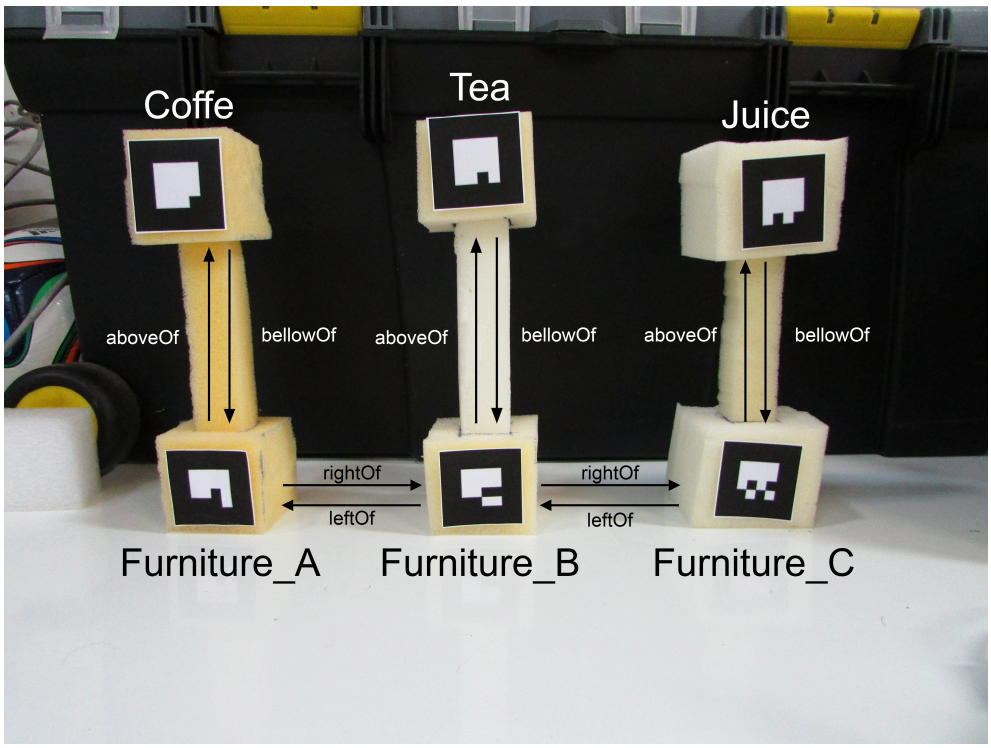


Figura 3.4: Ambiente observável. Os objetos que são *Drinks* estão sempre acima dos que são *Furniture* para simular uma situação real. As setas indicam as relações espaciais que cada objeto tem nesta configuração.

pode manipular os objetos *Drinks*, que sempre estão acima dos objetos *Furnitures*. Para demonstrar a capacidade de inferir informações, apenas os objetos *Furnitures* possuem as relações *leftOf* e *rightOf*. Desta forma, não há uma relação direta entre os *Drinks*.

O mapa semântico consiste de relações espaciais entre os objetos processadas a partir das informações obtidas pelo sensor Kinect. Nem todos os objetos aparecem na visão do robô, pois ele está sempre se movimentando. Por isso, o mapa não utiliza a posição e orientação de cada objeto, apenas as relações espaciais entre eles.

O comportamento do robô utiliza o mapa semântico para saber como chegar até o objeto especificado pelo usuário. Isto é feito procurando por uma sequência de objetos que o robô deve encontrar até chegar ao objeto alvo. Para cada relação espacial entre dois objetos, é associada uma ação que o robô deve executar para encontrar o próximo objeto. Ao encontrar o objeto especificado pelo usuário, o robô se locomove até ele, ajusta a sua posição e executa a ação de agarrar o objeto.

A seguir, são descritas as especificações de cada etapa da aplicação desenvolvida.

3.5.1 Arquivo OWL

As propriedades e classes desenvolvidas para a aplicação estão nas Tabelas 3.4 e 3.5. As classes *HumanScaleObject*, *string*, *VisualPerception*, *Drink* e *FurniturePiece* são oriundas do arquivo `knowrob.owl`. A propriedade *subClassOf* é utilizada para criar a hierarquia entre as classes.

Propriedade	Domínio	Intervalo	*Parente*
spatialRelation	*HumanScaleObject*	*HumanScaleObject*	
bellowOf			*spatialRelation*
aboveOf			*spatialRelation*
leftOf			*spatialRelation*
rightOf			*spatialRelation*
genericName	*HumanScaleObject*	*string*	
partOfMap	*Map*	*HumanScaleObject*	

Tabela 3.4: Propriedades utilizadas na aplicação. Além do domínio e intervalo, é apresentado o parente da propriedade, quando houver.

Os indivíduos criados pelo sistema de percepção podem ser instâncias das classes *Coffee*, *Tea*, *Juice* e *Furniture*. As classes que herdam de *VisualPerception* são utilizadas para diferenciar a origem destes indivíduos. Isto é necessário para identificar as partes do mapa, que são os indivíduos detectados pelo Kinect e os objetos que o robô está vendo em tempo real, que são indivíduos detectados pela câmera do robô. *DefaultObjectDetection* é utilizado para câmeras que não pertencem aos outros tipos de percepções.

3.5.2 Arquivo Prolog

No `KnowRob`, o conhecimento obtido dos arquivos OWL é armazenado como triplas [12]. Essas triplas possuem como elementos um sujeito, um predicado e um objeto. O arquivo `owl.pl` possui predicados que acessam essas triplas através de quetionamentos no Prolog. Estes predicados são *rdf_has/3 e rdf_triple/3.* O *rdf_triple/3,* além de retornar os resultados de informações não computáveis, retorna os resultados computados pelo sistema de percepção. É possível utilizar variáveis para encontrar o sujeito, predicado ou objeto de uma tripla específica.

Os predicados desenvolvidos para a aplicação são utilizados para criar e acessar as informações da base de conhecimento. A seguir está o código de cada predicado junto com a explicação da operação que eles fazem.

3.5.2.1 *perceptionFromTYPE/1*

Esse predicado verifica se um indivíduo é uma percepção da câmera TYPE, ou retorna os indivíduos que são perceção dessa câmera. TYPE pode ser Default, Kinect ou Nao.

```
perceptionFromTYPE(Obj) :-
  rdf_triple(rdf:type, Inst,knowrob:'TYPEObjectDetection'),
  rdf_triple(knowrob:'latestDetectionOfObject', Obj, Inst).
```

Classe	Propriedades	Recurso da propriedade
DefaultObjectDetection	*subClassOf*	*VisualPerception*
KinectObjectDetection	*subClassOf*	*VisualPerception*
NaoObjectDetection	*subClassOf*	*VisualPerception*
Coffee	*subClassOf*	*Drink*
	bellowOf	*HumanScaleObject*
	genericName	*HumanScaleObject*
Tea	*subClassOf*	*Drink*
	bellowOf	*HumanScaleObject*
	genericName	*HumanScaleObject*
Juice	*subClassOf*	*Drink*
	bellowOf	*HumanScaleObject*
	genericName	*HumanScaleObject*
Furniture	*subClassOf*	*FurniturePiece*
	aboveOf	*HumanScaleObject*
	leftOf	*HumanScaleObject*
	rightOf	*HumanScaleObject*
	genericName	*HumanScaleObject*
Map	*partOfMap*	*HumanScaleObject*

Tabela 3.5: Classes utilizadas nas aplicações. É apresentado as restrições de cada classe, com o nome da propriedade e a classe dos indivíduos relacionados.

3.5.2.2 *hasValue/2*

Utilizado para verificar se a variável já foi inicializada. O predicado *nonvar/1* determina se o argumento tem valor(es) fixo(s), ou seja, é não variante. *hasValue/2* é utilizado nos quetionamentos quando há necessidade de acessar os valores de uma variável. Quando o predicado não é usado, caso tente acessar uma variável sem valores fixos, é enviado um sinal de erro que encerra o programa.

```
hasValue(Term,True) :-
  (nonvar(Term) -> True = true ; True = false).
```

3.5.2.3 *createObjectPerception/5*

Baseado no predicado *create_object_perception/4* que já vem no `KnowRob`, esse predicado é responsável por criar as instâncias dos objetos gerados pelo sistema de percepção. Diferente da versão original, em vez de criar um novo indivíduo para cada nova percepção, ele apenas atualiza as propriedades quando o indivíduo já existe. Isso permite que o mapeamento e a busca sejam muito mais rápidos, pois a quantidade de indivíduos é reduzido e não há necessidade de verificar qual deles é a percepção mais recente do objeto.

Dentre as propriedades que os indivíduos têm, são atualizadas o tipo de percepção, a matriz homogênea com a pose do objeto e seu nome genérico. Indivíduos de um mesmo objeto possuem o mesmo nome genérico, o que torna possível a busca por objetos no mapa comparando com indivíduos de uma câmera diferente. É possível expandir o predicado para atualizar outras propriedades, como por exemplo, a cor e o formato do objeto.

```
createObjectPerception(ObjClass, ObjPose, PerceptionTypes,
ObjInst, ObjName) :-
  (owl_individual_of(ObjInst,ObjClass)) ->
    (create_perception_instance(PerceptionTypes, Perception),
    set_object_perception(ObjInst, Perception),
    set_perception_pose(Perception, ObjPose))
    ;
    (instanceFromClass(ObjClass, ObjInst),
    setGenericName(ObjName, ObjInst),
    create_perception_instance(PerceptionTypes, Perception),
    set_object_perception(ObjInst, Perception),
    set_perception_pose(Perception, ObjPose)).
```

3.5.2.4 *createMap/0*

Predicado que cria o mapa do ambiente. Como um objeto pode ter mais de um indivíduo, gerados por câmeras diferentes, os que são utilizados no mapeamento são relacionados ao indivíduo *mapa* pela propriedade *partOfMap*.

Os computáveis *toTheRightOf* e *inFrontOf-Generally* retornam como valores das variáveis todas as combinações possíveis para a relação espacial. A sequência de chamadas utilizadas para computar as informações garante que cada variável terá apenas um valor. Note que os computáveis utilizados diferem do nome das propriedades equivalentes, mostradas após o símbolo '%'. Isso acontece porque os eixos cartesianos da câmera são diferentes dos eixos utilizados pelo `KnowRob` para calcular as relações espaciais.

Este mapeamento é específico para a aplicação desenvolvida, e só funciona se houver 6 objetos diferentes na visão da câmera. Os objetos *Drinks* devem estar emcima dos objetos *Furniture,* como mostrado na Figura 3.4.

Para demonstrar a capacidade de inferir informações, apenas as instâncias da classe *Furniture* possuem as propriedades *leftOf* e *rightOf*. Assim, a única informação espacial que se tem de um objeto *Drink* é que está acima de um objeto *Furniture.*

Devido ao método de busca do Prolog [10], o mapeamento é relativamente lento. Para diminuir esse tempo, no começo foi garantido pelo predicado *alldif/1* que todas as variáveis possuem valores diferentes.

```
createMap :-
  alldif([ObjA, ObjB, ObjC, FurnA, FurnB, FurnC]),

  (rdf_triple(knowrob:'partOfMap',knowrob:'mapa',ObjA),
  rdf_triple(knowrob:'partOfMap',knowrob:'mapa',ObjB),
  rdf_triple(knowrob:'partOfMap',knowrob:'mapa',ObjC),
  rdf_triple(knowrob:'partOfMap',knowrob:'mapa',FurnA),
  rdf_triple(knowrob:'partOfMap',knowrob:'mapa',FurnB),
  rdf_triple(knowrob:'partOfMap',knowrob:'mapa',FurnC)),

  rdf_triple(knowrob:'inFrontOf-Generally',ObjA,ObjB),%leftOf
  rdf_triple(knowrob:'inFrontOf-Generally',ObjB,ObjC),%leftOf
  rdf_triple(knowrob:'inFrontOf-Generally',FurnA,FurnB),%leftOf
  rdf_triple(knowrob:'inFrontOf-Generally',FurnB,FurnC),%leftOf
  rdf_triple(knowrob:'toTheRightOf',ObjA,FurnA),%bellowOf
  rdf_triple(knowrob:'toTheRightOf',ObjB,FurnA),%bellowOf
  rdf_triple(knowrob:'toTheRightOf',ObjC,FurnA),%bellowOf
  rdf_triple(knowrob:'toTheRightOf',ObjA,FurnB),%bellowOf
  rdf_triple(knowrob:'toTheRightOf',ObjB,FurnB),%bellowOf
  rdf_triple(knowrob:'toTheRightOf',ObjC,FurnB),%bellowOf
  rdf_triple(knowrob:'toTheRightOf',ObjA,FurnC),%bellowOf
  rdf_triple(knowrob:'toTheRightOf',ObjB,FurnC),%bellowOf
  rdf_triple(knowrob:'toTheRightOf',ObjC,FurnC),%bellowOf

  rdf_assert(FurnA, knowrob:'leftOf',    FurnB),
  rdf_assert(FurnB, knowrob:'rightOf',   FurnA),
```

```
  rdf_assert(FurnB, knowrob:'leftOf', FurnC),
  rdf_assert(FurnC, knowrob:'rightOf', FurnB),
  rdf_assert(ObjA, knowrob:'bellowOf', FurnA),
  rdf_assert(FurnA, knowrob:'aboveOf', ObjA),
  rdf_assert(ObjB, knowrob:'bellowOf', FurnB),
  rdf_assert(FurnB, knowrob:'aboveOf', ObjB),
  rdf_assert(ObjC, knowrob:'bellowOf', FurnC),
  rdf_assert(FurnC, knowrob:'aboveOf', ObjC).
```

3.5.2.5 *findPath/4*

Predicado utilizado para encontrar a partir do mapa semântico um caminho entre o objeto alvo e o objeto que está na visão do robô. Ele retorna uma lista com a sequência de objetos que o robô deve procurar para chegar até o alvo e as propriedades que relacionam um objeto da lista com o próximo.

O método utilizado para encontrar o caminho é busca em grafos. Os objetos são nós e as propriedades são os arcos. Os termos *NextToFind \== Object* e *\+member(C,Visited)* garantem que o caminho não vai passar duas vezes pelo mesmo nó. É importante especificar que a propriedade deve ser do tipo *spatialRelation*. Caso não seja especificado, o programa irá testar todas as propriedades possíveis.

```
findPath(Subect,Object,ObjectsList,PropertiesList) :-
  path(Subect,Object,[],[],Objects,Properties),
  \+member(Subect,Objects),!,
  reverse(Objects,ObjectsList),
  reverse(Properties,PropertiesList).

path(Subect,Object,IObjectsList,IPropertiesList,
[Obj|IObjectsList],[Property|IPropertiesList]) :-
  rdf_has(Property, rdfs:subPropertyOf,
knowrob:'spatialRelation'),
  rdf_has(Subect,Property,Object).

path(Subect,Object,IObjectsList,IPropertiesList,OObjectsList,
OPropertiesList) :-
  rdf_has(Property, rdfs:subPropertyOf,
knowrob:'spatialRelation'),
  rdf_has(Subect,Property,NextToFind),
  NextToFind \== Object,
  \+member(NextToFind,IObjectsList),
  path(NextToFind,Object,[NextToFind|IObjectsList],
[Property|IPropertiesList],OObjectsList,OPropertiesList).
```

3.5.3 Interface com o usuário

Para que o usuário possa escolher o objeto que o robô deve manipular, foi implementado uma interface simples com um teclado. Teclas específicas foram associadas a quetionamentos que acessam o mapa semântico utilizando apenas informações simples. Além do nome genérico do objeto, é possível especificar informações que podem ser inferida observando o mapa. A lista de objetos e informações que podem ser utilizadas está a seguir:

- *Coffee*;
- *Tea*;
- *Juice*;
- *Drink* mais à esquerda;
- *Drink* no centro;
- *Drink* mais à direita;
- *Drink* acima do *Furniture* mais à esquerda;
- *Drink* acima do *Furniture* no centro;
- *Drink* acima do *Furniture* mais à direita.

3.5.4 Comportamento do robô

O comportamento do robô pode ser dividido em três partes: locomover até o objeto, centralizar o objeto na imagem e manipulação do objeto. Na locomoção até o objeto alvo, as listas de objetos e de propriedades encontradas com o predicado *findPath/4* são utilizadas para definir que ação o robô deve executar.

Foi feita uma relação direta entre as propriedades e as ações, como mostra a Tabela 3.6. Uma ação é concluída quando o próximo objeto da lista entra no campo de visão do robô. Também é possível encontrar objetos que estão na lista mas não são o próximo que deve ser encontrado. Nesse caso, os objetos que deveriam ser encontrados antes são retirados da lista.

Propriedade	Ação
bellowOf	$HeadPitch = 0.39rad$
aboveOf	$HeadPitch = 0.00rad$
leftOf	Locomoção para a esquerda
rightOf	Locomoção para a direita

Tabela 3.6: Relação entre a propriedades e ações que devem ser executadas para encontrar o próximo objeto. A locomoção para direita ou para a esquerda é feita com velocidade constante.

Quando o objeto alvo é detectado, o robô utiliza a posição e orientação encontrados em tempo real para se posicionar de uma forma que possa agarrá-lo. Ao mesmo tempo, a cabeça se movimenta para centralizar o objeto na imagem. Quando ele encontra uma posição válida, o robô então realiza a ação de agarrar o objeto.

3.6 Algoritmo de execução da aplicação

No Algoritmo 3.1 está a sequência de tarefas que são executadas pelo sistema para realizar aplicação. Antes de começar, é necessário inicializar os arquivos OWL e Prolog para acessar o conhecimento que não é obtido por computáveis. O mapeamento é feito antes de começar a busca para que o predicado *findPath/4* possa encontrar um caminho válido utilizando o mapa semântico. Por fim, e robô executa a manipulação do objeto alvo.

Algorithm 3.1 Execução da aplicação

1: Inicializa os arquivos OWL e Prolog;
2: Detecta os 6 objetos com o sensor Kinect;
3: Cria o mapa semântico;
4: Recebe o comando do usuário;
5: Infere qual é o objeto alvo;
6: Detecta que objeto a câmera do robô está vendo;
7: Cria as listas de objetos e propriedades que indicam um caminho para o objeto alvo;
8: **while** não é objeto alvo **do**
9: Executa a ação associada à primeira propriedade da lista até encontrar o próximo objeto;
10: **end while**
11: Locomove até 0.200m de distância do objeto alvo;
12: Ajusta a posição até achar uma solução para a cinemática inversa;
13: Move o braço até o lado do objeto alvo;

Capítulo 4

Experimentos e Resultados

4.1 Introdução

Para verificar o funcionamento da plataforma de pesquisa, foram feitos experimentos para testar o sistema de percepção e o controle do robô. No sistema de percepção foi analisado a capacidade de utilizar múltiplas câmeras e a precisão dos dados obtidos através dos marcadores. No controle do robô, cada tipo de ação utilizada foi testada primeiro em simulação e depois no robô real. Os testes da aplicação consistiram em manipular um objeto realizando todos os tipos de ações desenvolvidos e procurar por objetos em mapas organizados com diferentes configurações. Por razões descritas na Seção 4.3.1, não foi possível testar a aplicação de forma completa. No lugar do robô, foi utilizado uma câmera externa controlada manualmente pelo operador para procurar por objetos.

4.2 Sistema de percepção

Para testar o sistema de percepção, foi verificado a precisão dos dados obtidos e a capacidade de utilizar mais de uma câmera ao mesmo tempo. A percepção utilizando duas câmeras pode ser vista na Figura 4.1. Utilizando o `ARToolKit` foi possível obter as informações de ambas as câmeras ao mesmo tempo. Uma delas visualiza todos os objetos para fazer o mapeamento, enquanto que a outra é utilizada para saber a distância relativa entre os objetos e o robô. Cada marcador possui duas percepções, diferenciadas pelo identificador de cada câmera.

Quatro pequenos experimentos foram feitos para verificar a precisão dos dados obtidos dos marcadores. Os experimentos consistiram em calcular a distância, o ângulo e os deslocamentos nos eixos X e Y do marcador mais à direita em relação ao marcador mais à esquerda em diferentes configurações. Os dados obtidos estão na Tabela 4.1 e foram comparados com as medidas reais entre os marcadores. Cada marcador utilizado nos experimentos tem $0.102m$ de lado.

No experimento 1, dois marcadores foram dispostos lado a lado com os eixos em paralelo, como mostrado na Figura 4.2. A distância real medida entre os centros dos marcadores foi de $0.211m$. Anasilando os dados da Tabela 4.1, a diferença entre a medida real e a feita pelo sistema foi de

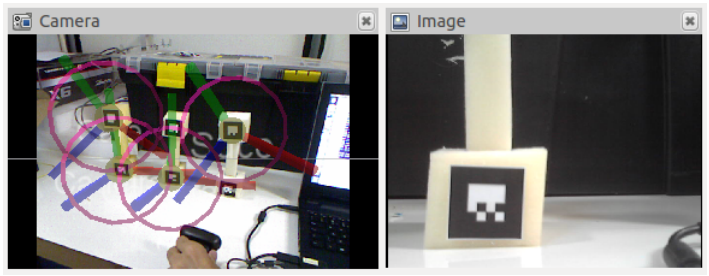


Figura 4.1: Uso de múltiplas câmeras. As informações de cada câmera podem ser obtidas ao mesmo tempo. Os dados de um mesmo marcador são diferenciados pelo identificador da câmera usada.

Experimento	Distância(m)	Ângulo(rad)	X(m)	Y(m)
1	0.201	0.005	0.201	0.001
2	0.177	1.585	-0.003	0.177
3	0.226	-0.010	0.226	-0.002
4	0.336	0.965	0.191	0.276

Tabela 4.1: Dados relativos do marcador da direita em relação ao da esquerda obtidos dos experimentos 1, 2, 3 e 4 da detecção de marcadores.

$0.010m$, um erro aceitável para as ações realizadas pelo robô.

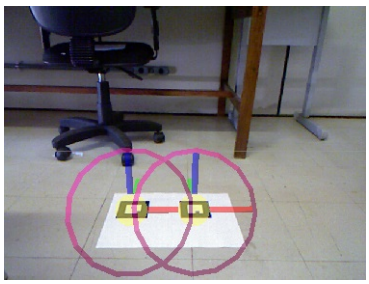


Figura 4.2: Configuração do experimento 1 da detecção de marcadores. Os marcadores estão alinhados com os eixos em paralelo.

No experimento 2 mostrado na Figura 4.3, partindo da configuração do experimento 1, o marcador da esquerda foi rotacionado aproximadamente 90^o no sentido horário do eixo Z. Como mostrado na Tabela 4.1, houve uma mudança do ângulo entre os marcadores para aproximadamente $90^o(1.57rad)$ e a distância a ser percorrida passou do eixo X para o eixo Y. No caso do experimento 3 mostrado na Figura 4.4, onde o marcador da direita foi rotacionado, não houve mudança no ângulo ou no eixo referente à distância a percorrer. Isso ocorreu porque os dados foram obtidos utilizando o marcador da esquerda como referência e não houve mudanças na sua orientação comparado ao experimento 1.

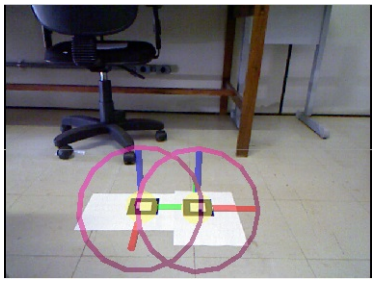


Figura 4.3: Configuração do experimento 2 da detecção de marcadores. O marcador da esquerda foi rotacionado 90^o no sentido horário do eixo Z.

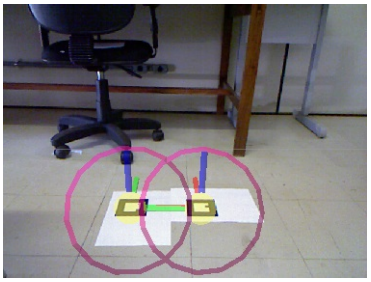


Figura 4.4: Configuração do experimento 3 da detecção de marcadores. O marcador da direita foi rotacionado 90^o no sentido anti horário do eixo Z.

No experimento 4, partindo da configuração do experimento 1, o marcador da direita foi transladado na direção do eixo Y em uma distância igual à altura do papel, que é aproximadamente $0.297m$. Como observado na Tabela 4.1, a distância no eixo X continuou com o mesmo erro em relação à medida real. A diferença entre a medida real e a obtida no eixo Y foi de aproximadamente $0.020m$, um valor maior em comparação ao eixo X. Como causa do aumento do erro pode-se apontar o aumento da distância do marcador em relação à câmera ou a redução do tamanho do marcador visto na imagem.

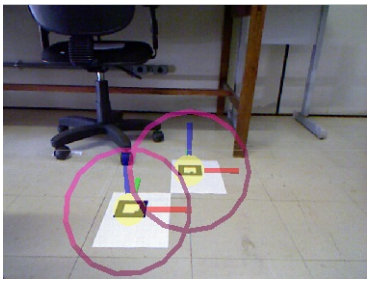


Figura 4.5: Configuração do experimento 4 da detecção de marcadores. O marcador da direita foi deslocado na direção do eixo Y.

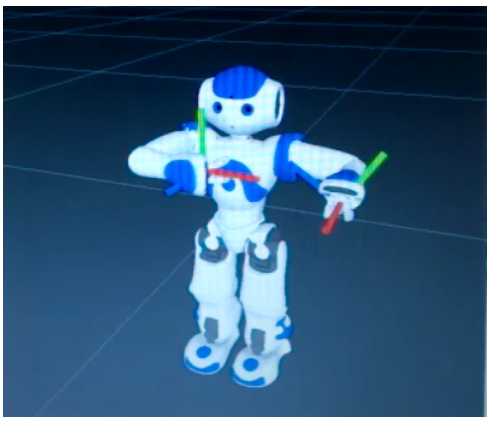


Figura 4.6: Experimento em simulação. A solução numérica do `IKFast` foi utilizada para os braços realizarem trajetórias específicas..

(a) (b)

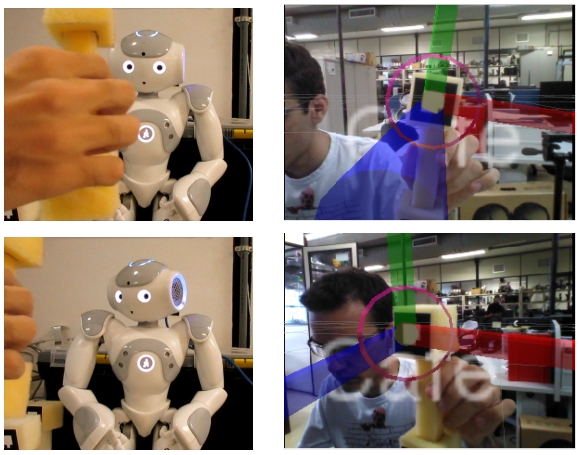
Figura 4.7: Posições válidas para manipular um objeto (a) na frente do ombro e (b) na frente do botão de ligar do robô.

4.3 Controle do robô

Para verificar a realização das ações no robô, primeiro foram realizados experimentos em simulação. Foi utilizado a ferramenta *rviz* disponível no ROS junto com o pacote `nao_description`, que possui a cinemática e um modelo do robô Nao. A simulação foi utilizada para evitar que o robô realizasse movimentos que pudessem danificá-lo. As ações foram realizadas no robô apenas quando a simulação não demonstrava nenhum risco.

Um experimento para testar a solução numérica do `IKFast` em simulação é mostrado na Figura 4.6. O objetivo era verificar se existia uma solução para posições na frente do ombro e em torno do botão central do robô. Para o braço direito, foi calculado uma trajetória circular em torno do botão central. A orientação considerada foi o eixo X da mão orientado para a esquerda, na visão do robô. O braço esquerdo realizou uma trajetória retilínea ao longo do eixo X, que está orientado para a frente. Em ambos os casos os eixos Y e Z podiam variar. O que se observou foi a capacidade de posicionar os braços em regiões superiores, tanto na frente do ombro como perto do botão. Em regiões inferiores só foram encontradas poucas orientações em que existia uma solução válida. Exemplos de posições válidas no robô real são mostrados na Figura 4.7.

Nos experimentos do movimento da cabeça, os ângulos dos atuadores do pescoço são ajustados



(a) (b)

Figura 4.8: Movimento da cabeça (a) do robô e (b) na visão do robô. O objeto é centralizado para que não saia do campo de visão.

até que o marcador esteja em uma faixa de $5.156^o(0.09rad)$ para mais e para menos em relação ao centro da câmera. Essa precisão é suficiente para que o marcador continue dentro do campo de visão do robô, considerando que a velocidade com que o marcador se movimenta é menor ou igual que a velocidade dos atuadores. Caso a velocidade seja maior, após um tempo o marcador sai do campo de visão. A realização do movimento da cabeça pode ser vista na Figura 4.8.

4.3.1 Limitações

Como o controle desenvolvido considera todas as soluções dentro dos limites dos atuadores do robô, é possível que as soluções para duas posições seguidas sejam diferentes. Além disso, o ângulo obtido para os *links* *l_gripper* e *r_gripper* não são levados em consideração para verificar os limites, pois o robô possui apenas 5 DoF em cada braço. Por conta desses dois fatores juntos, há casos em que uma solução é considerada válida, mesmo que os dedos não estejam apontados para o objeto, como mostrado na Figura 4.9. Também pode ocorrer do ângulo para o atuador do pulso está fora dos limites das juntas.

Figura 4.9: Falso positivo para as soluções do `IKFast`. O ângulo obtido para os *links* *l_gripper* e *r_gripper* não é levado em consideração para verificar os limites das soluções válidas.

Não foi possível realizar o movimento de fechar a mão em torno do objeto. Para isso, seria

necessário uma condição que indicasse quando o braço alcança o objeto. O único *feedback* obtido dos sensores do robô através do ROS são os ângulos de cada atuador. Cada ângulo obtido tem uma precisão de aproximadamente $5.156^o(0.09rad)$ para mais e para menos, o que torna o seu uso inviável quando consiredado o erro acumulado dos cinco atuadores.

Também não foi possível controlar corretamente a posição do robô quando ele se locomove de um ponto a outro, pois não há *feedback* da localização do robô no ambiente. Além disso, as funções para locomoção do `Naoqi` só param o movimento quando o robô está em uma posição estável. Assim, há casos em que o robô ultrapassa a posição onde ele deveria parar. Como alternativa foi utilizado como *feedback* a posição e orientação até o marcador obtida pela câmera, embora ainda seja necessário fazer ajustes para encontrar uma posição na qual seja possível manipular o objeto. Nos experimentos realizados, foi encontrado que parar a uma distância de $0.200m$ do marcador é ideal para conseguir manipular os objetos.

Nos últimos experimentos, o robô começou a apresentar problemas na movimentação. Os movimentos dos braços e da cabeça não apresentaram problemas, mas o robô não conseguia realizar a locomoção. O problema observado ocorreu da seguinte forma. Após levantar o primeiro pé, o robô para nesta posição. É possível enviar outros comandos mesmo após parar a locomoção e o notebook continua recebendo as informações da câmera. Em experimentos passados, foi observado que ao retirar o robô do chão durante a locomoção ele parava de se mover. Colocado novamente no chão, ele voltava a se mover. Acredita-se que os dados dos sensores de pressão nos pés são utilizados pelas funções do `Naoqi` para realizar essa ação como medida de segunrança. Se este for o caso, o mal funcionamento dos sensores pode fazer o sistema considerar que o robô não está no chão e por isso a locomoção para. Não foi verificado o funcionamento dos sensores de pressão.

4.4 Aplicação

A aplicação desenvolvida permitiu que o robô buscasse e manipulasse os objetos que compõem o ambiente simplificado. Os objetos utilizados, mostrados na Figura 4.10, foram feitos de espuma, pois os atuadores do robô não possuem torque suficiente para levantar objetos pesados.

Foram feitos cubos com tamanho suficiente para colar os marcadores de $0.040m$ de lado. As hastes possuem altura de aproximadamente $0.080m$ e lado de $0.015m$. O lado da haste foi o maior possível para que o objeto ficasse em pé e fosse fino o suficiente para o robô poder agarrar. Quando ligados pela haste, o cubo inferior representa um objeto *Furniture*, enquanto que o cubo superior representa um objeto *Drink*. A relação entre os marcadores e os objetos que eles representam está na Tabela 4.2.

Os experimentos da aplicação podem ser divididos em duas partes. A manipulação do objeto alvo e a busca pelo objeto alvo. O experimento de manipulação é mostrada na Figura 4.11. Ao detectar o objeto, o robô se locomoveu até uma distância de $0.200m$ do marcador. Ele então ajustou a sua posição até que a mão pudesse alcançar o objeto e posicionou a sua mão ao lado do objeto. Não foi possível finalizar com a ação de fechar a mão ao redor do objeto devido à falta de precisão dos ângulos obtidos pelos sensores do robô.

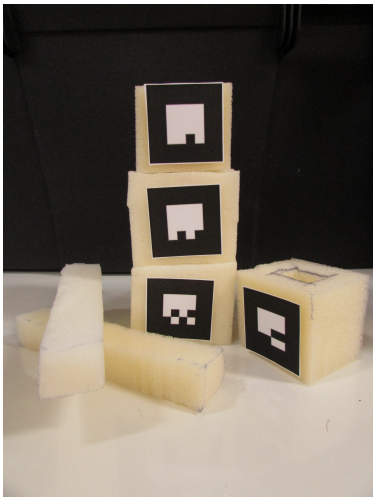
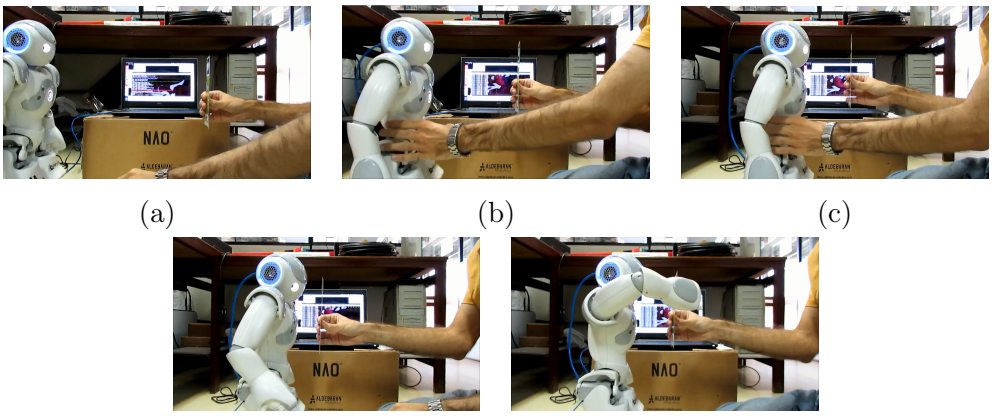


Figura 4.10: Objetos utilizados nos experimentos. Eles são feitos de espuma e possuem base e topo largos para colar os marcadores e haste fina para que o robô possa manipulá-los.

Marcador	Objeto
1	*Coffee*
2	*Tea*
5	*Juice*
43	*Furniture_A*
57	Furniture_B
83	Furniture_C

Tabela 4.2: Relação entre marcadores e os objetos que representam. Cada marcador representa um número em binário, que são representados em número decimal na tabela.



(a) (b) (c)

(d) (e)

Figura 4.11: Manipulação de um objeto. (a) Ao detectar o objeto, o robô (b) se locomove (c) até uma distância de $0.200m$ do marcador. (d) Ele se ajusta para encontrar uma posição válida e (e) extende seu braço até o objeto.

Os experimentos para procurar objetos foram feitos inicialmente com uma câmera controlada manualmente, simulando as ações que o robô deve realizar. Para mover a câmera até o objeto especificado pelo usuário a partir do objeto que ela está vendo, foi utilizado as informações em tempo real obtidas pela câmera e o caminho criado pelo predicado *findPath/4*. O objeto alvo é encontrado procurando pela sequência de objetos intermediários que formam o caminho.

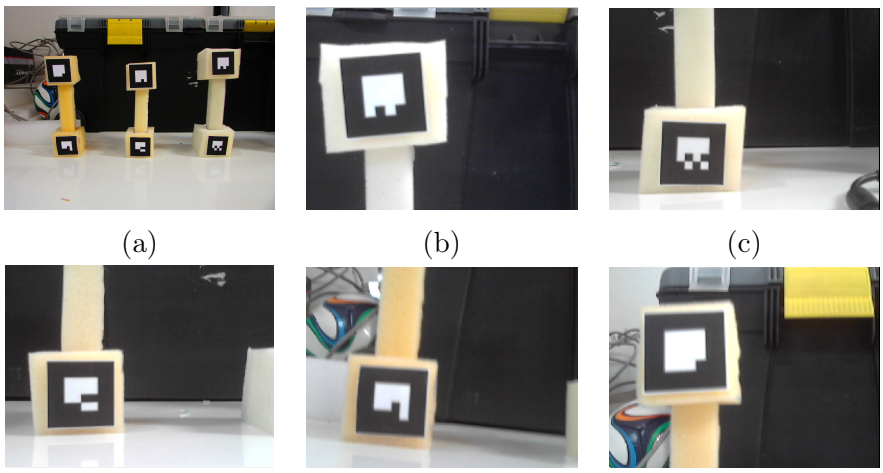
Foram realizados seis experimentos para testar a busca. Neles, a câmera precisava encontrar o objeto alvo de acordo com as instruções "Encontrar objeto tipo *Coffee*" e "Encontrar *Drink* mais à esquerda". Para cada instrução, foram testadas três formas diferentes de organizar os marcadores, considerando que os objetos tipo *Drink* estão sempre acima dos objetos tipo *Furniture*.

No experimento mostrado na Figura 4.12, o objeto tipo *Coffee* deve ser encontrado a partir do objeto inicial, que é do tipo *Juice*. Como mostrado na Figura 4.12.a, *Coffee* se encontra no canto superior esquerdo, enquanto que *Juice* está no canto superior direito. Considerando as propriedades das classes na Tabela 3.5, a câmera deve encontrar os objetos *Furniture_*C, *Furniture_B* e *Furniture_A* nesta ordem para encontrar *Coffee*. Não é possível utilizar o caminho pelo objeto tipo *Tea* porque as classes que herdam de *Drink* não possuem a propriedade *leftOf*.

As listas de objetos e propriedades obtidas através de *findPath/4* estão na Tabela 4.3. A sequência de objetos a serem encontrados, de cima para baixo, está de acordo com a sequência correta. A partir das propriedades que associam estes objetos, a câmera foi movimentada de acordo com as ações equivalentes na Tabela 3.6. Como resultado, partindo do objeto tipo *Juice*, mostrado na Figura 4.12.b, foram encontrados os marcadores das Figuras 4.12.c, 4.12.d, 4.12.e e 4.12.f. De acordo com a Tabela 4.2, estes marcadores representam os objetos da sequência correta, sendo que o último é o objeto tipo *Coffee*.

Nos outros experimentos, mostrados nas Figuras 4.13, 4.14, 4.15, 4.16 e 4.17, também foi possível encontrar o objeto alvo a partir do objeto inicial seguindo um caminho válido. As listas de objetos e propriedades obtidas através de *findPath/4* estão nas Tabelas 4.4, 4.5, 4.6, 4.7 e 4.8, respectivamente.

No período que os experimentos de busca foram realizados, o robô já apresentava problemas na locomoção. Assim, não foi realizado um teste completo com a busca e a manipulação dos objetos. Entretanto, os testes individuais foram suficientes para validar a aplicação e o algoritmo pode ser usado como base para realizar tarefas mais complexas em ambientes dinâmicos.



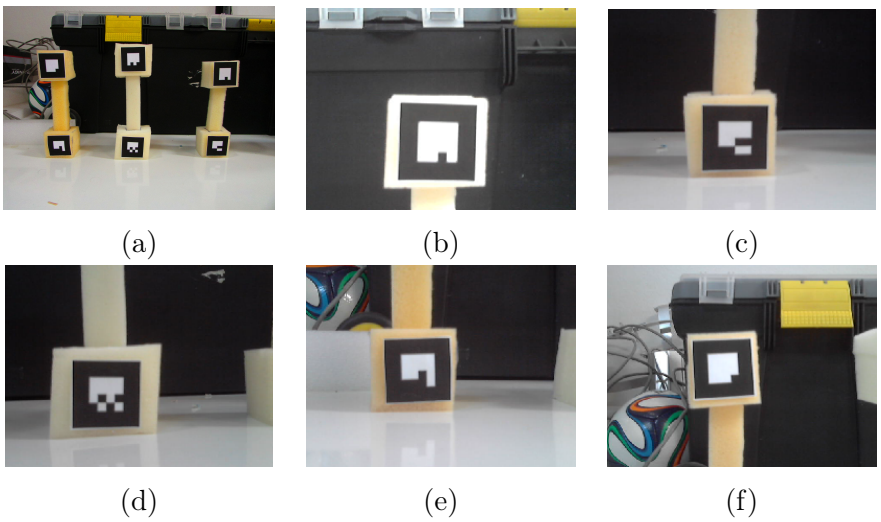
(a) (b) (c)

(d) (e) (f)

Figura 4.12: Primeiro experimento de encontrar o objeto tipo *Coffee*. (a) Organização dos objetos utilizada. (b) *Juice* é o primeiro objeto visto pela câmera. (c) *Furniture_ C*, (d) *Furniture_ B* e (e) *Furniture_ A* são os objetos intermediários encontrados, nesta ordem. (f) *Coffee* é o objeto a ser encontrado. As listas de objetos e propriedades do experimento se encontram na Tabela 4.3.

Propriedade	Objeto
	Juice
bellowOf	*Furniture_ C*
leftOf	*Furniture_ B*
leftOf	*Furniture_ A*
aboveOf	*Coffee*

Tabela 4.3: Listas de objetos e propriedades do primeiro experimento de encontrar o objeto tipo *Coffee*. O primeiro objeto da lista é primeiro objeto visto pela câmera, por isso não há propriedade entre outro objeto e ele. A propriedade relaciona o objeto anterior da lista com o próximo.



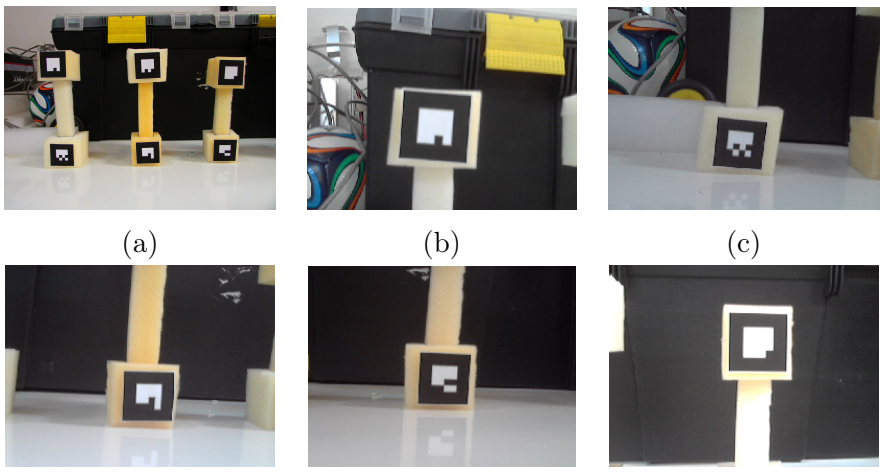
(a) (b) (c)

(d) (e) (f)

Figura 4.13: Segundo experimento de encontrar o objeto tipo *Coffee*. (a) Organização dos objetos utilizada. (b) *Tea* é o primeiro objeto visto pela câmera. (c) *Furniture_*B, (d) *Furniture_ C* e (e) *Furniture_ A* são os objetos intermediários encontrados, nesta ordem. (f) *Coffee* é o objeto a ser encontrado. As listas de objetos e propriedades do experimento se encontram na Tabela 4.4.

Propriedade	Objeto
	Tea
bellowOf	*Furniture_ B*
leftOf	*Furniture_ C*
leftOf	*Furniture_ A*
aboveOf	*Coffee*

Tabela 4.4: Listas de objetos e propriedades do segundo experimento de encontrar o objeto tipo *Coffee*. O primeiro objeto da lista é primeiro objeto visto pela câmera, por isso não há propriedade entre outro objeto e ele. A propriedade relaciona o objeto anterior da lista com o próximo.



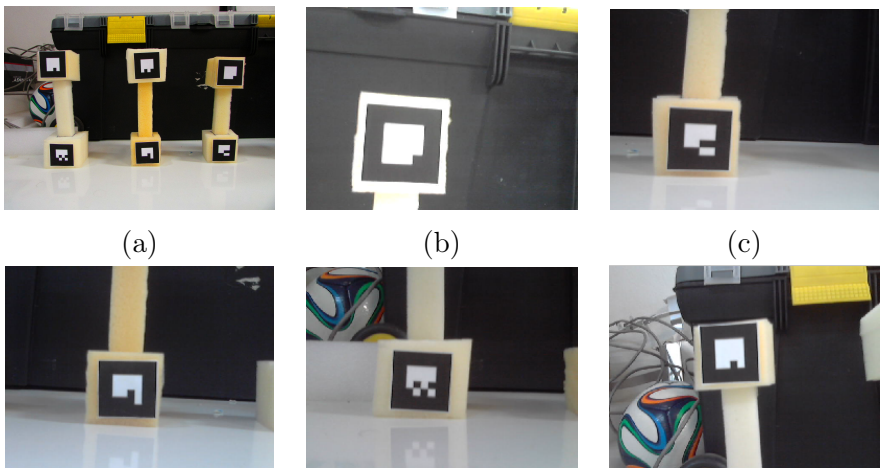
(a) (b) (c)

(d) (e) (f)

Figura 4.14: Terceiro experimento de encontrar o objeto tipo *Coffee*. (a) Organização dos objetos utilizada. (b) *Tea* é o primeiro objeto visto pela câmera. (c) *Furniture_C*, (d) *Furniture_A* e (e) *Furniture_B* são os objetos intermediários encontrados, nesta ordem. (f) *Coffee* é o objeto a ser encontrado. As listas de objetos e propriedades do experimento se encontram na Tabela 4.5.

Propriedade	Objeto
	Tea
bellowOf	*Furniture_C*
rightOf	*Furniture_A*
rightOf	*Furniture_B*
aboveOf	*Coffee*

Tabela 4.5: Listas de objetos e propriedades do terceiro experimento de encontrar o objeto tipo *Coffee*. O primeiro objeto da lista é primeiro objeto visto pela câmera, por isso não há propriedade entre outro objeto e ele. A propriedade relaciona o objeto anterior da lista com o próximo.



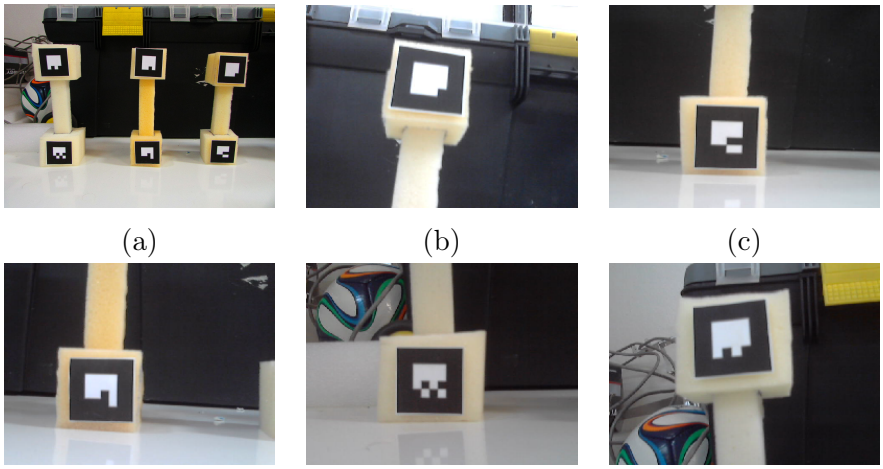
(a) (b) (c)

(d) (e) (f)

Figura 4.15: Primeiro experimento de encontrar o *Drink* mais à esquerda. (a) Organização dos objetos utilizada. (b) *Coffee* é o primeiro objeto visto pela câmera. (c) *Furniture_*B, (d) *Furniture_A* e (e) *Furniture_C* são os objetos intermediários encontrados, nesta ordem. (f) *Tea* é o *Drink* mais à esquerda. As listas de objetos e propriedades do experimento se encontram na Tabela 4.6.

Propriedade	Objeto
	Coffee
bellowOf	*Furniture_B*
leftOf	*Furniture_A*
leftOf	*Furniture_C*
aboveOf	*Tea*

Tabela 4.6: Listas de objetos e propriedades do primeiro experimento de encontrar o *Drink* mais à esquerda. O primeiro objeto da lista é primeiro objeto visto pela câmera, por isso não há propriedade entre outro objeto e ele. A propriedade relaciona o objeto anterior da lista com o próximo.



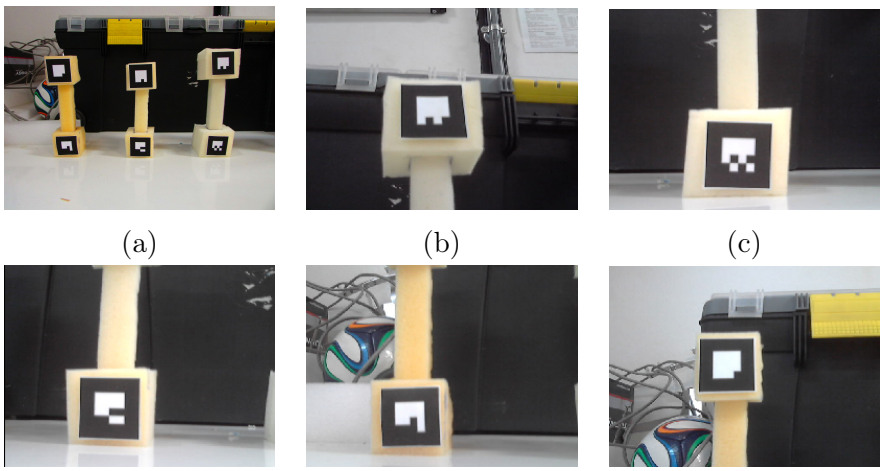
(a) (b) (c)

(d) (e) (f)

Figura 4.16: Segundo experimento de encontrar o *Drink* mais à esquerda. (a) Organização dos objetos utilizada. (b) *Coffee* é o primeiro objeto visto pela câmera. (c) *Furniture_*B, (d) *Furniture_ A* e (e) *Furniture_ C* são os objetos intermediários encontrados, nesta ordem. (f) *Juice* é o *Drink* mais à esquerda. As listas de objetos e propriedades do experimento se encontram na Tabela 4.7.

Propriedade	Objeto
	Coffee
bellowOf	*Furniture_ B*
leftOf	*Furniture_ A*
leftOf	*Furniture_ C*
aboveOf	*Juice*

Tabela 4.7: Listas de objetos e propriedades do segundo experimento de encontrar o *Drink* mais à esquerda. O primeiro objeto da lista é primeiro objeto visto pela câmera, por isso não há propriedade entre outro objeto e ele. A propriedade relaciona o objeto anterior da lista com o próximo.



(a) (b) (c)

(d) (e) (f)

Figura 4.17: Terceiro experimento de encontrar o *Drink* mais à esquerda. (a) Organização dos objetos utilizada. (b) *Juice* é o primeiro objeto visto pela câmera. (c) *Furniture_C*, (d) *Furniture_B* e (e) *Furniture_A* são os objetos intermediários encontrados, nesta ordem. (f) *Coffe* é o *Drink* mais à esquerda. As listas de objetos e propriedades do experimento se encontram na Tabela 4.8.

Propriedade	Objeto
	Juice
bellowOf	*Furniture_C*
leftOf	*Furniture_B*
leftOf	*Furniture_A*
aboveOf	*Coffee*

Tabela 4.8: Listas de objetos e propriedades do terceiro experimento de encontrar o *Drink* mais à esquerda. O primeiro objeto da lista é primeiro objeto visto pela câmera, por isso não há propriedade entre outro objeto e ele. A propriedade relaciona o objeto anterior da lista com o próximo.

Capítulo 5

Conclusões

O objetivo deste projeto é validar o uso de raciocínio formal para buscar e manipular objetos em um ambiente simplificado utilizando o robô humanóide Nao. Além do robô, foi utilizado um sensor Kinect para interagir e obter informações do ambiente. O comportamento baseado no pacote `KnowRob` foi capaz de combinar e recuperar de forma inteligente as informações armazenadas e prover informações em um nível intermediário que humanos e máquinas possam entender.

O desenvolvimento é dividido em três partes: sistema de percepção, controle do robô e o código específico da aplicação. O sistema de percepção utiliza as informações dos marcadores detectados para criar indivíduos na base de conhecimento. Dependendo de como os indivíduos serão utilizados, as suas informações antigas podem permanecer na base de conhecimento ou serem apagadas. Para o controle do robô, foram utilizadas as funções do `Naoqi` para a locomoção e desenvolvidos os movimentos dos braços e da cabeça. Foi necessário adicionar um *link* em cada braço para utilizar a solução númerica do pacote `IKFast`. Além disso, para centralizar o objeto com a câmera, as coordenadas do objeto foram transformadas para coordenadas da base do robô.

Como método padrão para desenvolver as aplicações que usam a plataforma de pesquisa, é necessário criar arquivos OWL e Prolog específicos da aplicação e desenvolver a interface com o usuário e o comportamento do robô. Na aplicação de procurar e agarrar objetos, foram criados classes, propriedades e predicados para mapear o ambiente e encontrar um caminho que leve até o objeto especificado pelo usuário. A sequência de ações que o robô executa estão de acordo com as propriedades que relacionam os objetos do caminho.

Os experimentos feitos durante o projeto serviram para verificar o funcionamento de todos os módulos da plataforma de pesquisa e validar a aplicação desenvolvida. Eles mostraram que o uso de marcadores é uma maneira simples e robusta de obter informações precisas sobre objetos no mundo real, além da capacidade de utilizar múltiplas câmeras para atualizar a base de conhecimento. No controle do robô, foi possível posicionar os braços em posições específicas e centralizar os marcadores na imagem da câmera com a movimentação da cabeça. Entretanto, houveram limitações na locomoção e movimento dos braços.

Devido às limitações do robô e o mal funcionamento nos últimos experimentos, nos experimentos da aplicação só foi possível testar de forma independente a busca e a manipulação dos

objetos. Utilizando as informações do sistema de percepção, foi possível manipular objetos, mas sem finalizar com o movimento de fechar a mão ao redor do objeto. Os experimentos da busca mostraram a capacidade do sistema de seguir diferentes instruções do usuário e encontrar o objeto alvo utilizando o conhecimento adquirido e adaptando as ações para diferentes circunstâncias.

A integração de módulos através do ROS foi fundamental para o desenvolvimento deste projeto. O uso de tópicos e serviços permitiu a comunicação entre nós com funcionalidades distintas, como o sistema de percepção e o controle do robô. Além disso, mesmo que a aplicação tenha sido simplificada, foi possível mostrar a capacidade do uso de representação de conhecimento e raciocínio formal para realizar uma atividade comum para seres humanos e uma comunicação de mais alto nível entre a pessoa e o robô.

5.1 Perspectivas Futuras

Este projeto serviu como validação para utilizar raciocínio formal no desenvolvimento de tarefas *pick-and-place* e manipulações mais complexas em ambientes dinâmicos. Aperfeiçoando cada módulo separadamente, é possível aumentar a quantidade de informações na base de conhecimento e ampliar a faixa de aplicações que é possível desenvolver.

Uma das principais mudanças que podem ser feitas é inserir na base de conhecimento as ações que o robô pode realizar. Adicionando restrições de quando e como uma ação deve ser utilizada, o robô pode inferir de forma autônoma que ações ele deve realizar para alcançar o objetivo da tarefa. Além disso, melhorar o uso da cinemática inversa e usar planejamento de trajetória permitirá aumentar a quantidade de ações que o robô pode fazer.

No sistema de percepção, a detecção de marcadores pode ser substituída por um algoritmo de visão computacional para obter mais informações sobre os objetos, como cor e formato. Essas informações podem ser utilizadas para classificar objetos que não são conhecidos previamente e encontrar objetos pertencentes a uma mesma classe, mesmo que algumas de suas propriedades sejam diferentes.

Um passo importante para criar aplicações mais complexas é o desenvolvimento da capacidade de entender linguagem natural. Isso facilitará a comunicação das pessoas com o robô e permitirá também o uso de comandos de voz. Além disso, para se locomover em ambientes de larga escala, é necessário a realização de um mapeamento híbrido com atualização global e local. Isso significa que o robô atualiza separadamente as informações dos objetos com os quais está interagindo ou que estão próximos e é capaz de perceber as mudanças feitas no ambiente.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] TENORTH, M. Knowledge processing for autonomous robots. *Technische Universität München*, 2011.

[2] KUNZE, L. et al. Searching objects in large-scale indoor environments: A decision-theoretic approach. *International Conference on Robotics and Automation*, p. 4385 – 4390, 2012.

[3] TENORTH, M.; BEETZ, M. Knowrob – a knowledge processing infrastructure for cognition-enabled robots. part 1: The knowrob system. *International Journal of Robotics Research (IJRR)*, v. 32, n. 5, p. 566 – 590, April 2013.

[4] KATO, H.; BILLINGHURST, M. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. *2nd International Workshop on Augmented Reality (IWAR 99)*, 1999.

[5] ROS - Robot Operating System. [S.l.]: Open Source Robotics Foundation, 2007. `http://www.ros.org`.

[6] ROBOT NAO. [S.l.]: Aldebaran.

[7] KINECT for Xbox 360. [S.l.]: Microsoft Corp. Redmond WA.

[8] SHOTTON, J. et al. Real-time human pose recognition in parts from single depth images. *Microsoft Research Cambridge & Xbox Incubation.*

[9] RUDOLPH, S. Foundations of description logics. *Karlsruhe Institute of Technology*, 2011.

[10] BLACKBURN, P.; BOS, J.; STRIEGNITZ, K. *Learn Prolog Now!* [S.l.]: College Publications, 2006. (Texts in Computing, v. 7).

[11] OWL - Web Ontology Language. [S.l.]: W3C, 2004. `http://www.w3.org/TR/owl-ref/`.

[12] TENORTH, M.; BEETZ, M. Knowrob – knowledge processing for autonomous personal robots. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[13] SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. Robot modeling and control. *JOHN WILEY & SONS, INC.*, p. 65 – 87, 2005.

[14] JOHNSON, D.; BERTHIAUME, C.; WITKOWSKI, B. *ARToolkit Patternmaker*. `http://www.cs.utah.edu/gdc/projects/augmentedreality/download.html`.

ANEXOS

I. DESCRIÇÃO DO CONTEÚDO DO CD

O CD segue a seguinte estrutura:

- leiame.pdf : Essa descrição do CD
- Trabalho_de_Graduação.pdf : Versão digital do trabalho de graduação
- Resumo_e_Palavras_Chave.pdf : O resumo e as palavras chave do trabalho de graduação
- \videos
  - Experimento_Coffee_1.mp4 : Primeiro experimento para o comando "Encontrar objeto tipo *Coffee*".
  - Experimento_Coffee_2.mp4 : Segundo experimento para o comando "Encontrar objeto tipo *Coffee*".
  - Experimento_Coffee_3.mp4 : Terceiro experimento para o comando "Encontrar objeto tipo *Coffee*".
  - Experimento_Left_1.mp4 : Primeiro experimento para o comando "Encontrar *Drink* mais à esquerda".
  - Experimento_Left_2.mp4 : Segundo experimento para o comando "Encontrar *Drink* mais à esquerda".
  - Experimento_Left_3.mp4 : Terceiro experimento para o comando "Encontrar *Drink* mais à esquerda".
  - Manipulação.mp4 : Experimento de manipulação de objetos realizando todas as ações.
  - Posicionamento_da_Mão.mp4 : Compilação de experimentos do posicionamento da mão perto do objeto.